菱木研二　斉藤　直・著

MicrosoftC Ver5.1

マイティ リファレンス

The Mighty References

ラジオ技術社

菱木研二　斉藤　直・著

MicrosoftC Ver5.1
マイティ リファレンス
The Mighty References

ラジオ技術社

【はじめに】

ここ数年、C言語は大変なブームになっています。ブームになるだけあって、C言語にはプログラマを魅了する数々の特徴があります。しかし、いざ使ってみようとしても、マニュアルや入門書だけで自由に使いこなせるようにはなりません。また、ただ沢山の資料を揃えればよいというものでもありません。

C言語を使いこなすには、まず美しいC言語の世界を理解することが必要です。そして、使いこなすための「ポイント」をおさえなければなりません。

本書では、MS-DOSで最もポピュラーなCコンパイラであるMicrosoftCバージョン5.1を通して、Cコンパイラを使いこなすためのポイントをまとめてみました。

ポイントとは言っても、その内容たるや、あまりにも広く深いものです。そのために、限られた誌面では、残念ながらすべての人に完全に理解していただけるような記述はできません。そこで本書は、次のような読者層を主な対象としています。

★MS-DOSやC言語に関する基礎的な知識があり、C言語以外の言語(BASIC、Pascalなど)でプログラミングをしたことがある。入門書を買ってC言語を勉強したが、現実的なプログラムの作り方がわからない。

★C言語を使用してプログラムを作成できるが、どうも、C言語を征服した気分になれない。

★C言語を使用して実際にソフトウェアを作成しているが、MicrosoftCバージョン5.1になって拡張された部分を有効に使っていないような気がする。

すなわち、MS-DOSやC言語に関するごく基本的な知識を持っている方を対象と考えています。したがって、C言語の言語仕様などについては特にページをさいてはいません。

全体の構成は、次のように2部からなっています。

【第1部】

●第1章

MicrosoftCはMS-DOSで動作しているという観点からC言語というものを見直すことによって、Cプログラミングの注意や基礎知識を読物風にまとめてみました。またMicrosoftC以外にも、最近のCコンパイラの話題を追ってみました。MS-DOSやマシンのデータブックを参照しながら読めば、いっそう詳しく理解できるでしょう。

●第2章

MicrosoftCバージョン5.1のインストール、コンパイル、リンクなどのオプションや動作などについて解説しました。バージョンによる違いやCodeViewの使用例についても記述しました。

●第3章

サンプルプログラムを通して、各関数の使い方などのポイントを解説しました。リファレンスと違って、関数はひとまとめのグループとして使い方を解説してあります。関数の有機的な使い方がわかるように記述しました。

【第2部】

いわゆるリファレンスです。MicrosoftCバージョン5.1に付属しているリファレンスは大変良くできているのですが、実際にユーザーが使用するときの要望とは必ずしも一致していないと思われる部分もあります。そこで本書では、関数をグループに分けて、独自の構成をとりました。間違えやすいポイントと、適切なサンプルプログラムを記述することにより、理解しやすい、使いやすい構成としました。

MicrosoftCは今回のバージョンアップによってOS/2でも使用可能となりましたが、本書ではMS-DOSで使用することを前提に記述してあります。OS/2に関する大部分は省略されています。また、使用マシンはPC-9801を前提としていますが、第2部ではAX専用関数についても解説しています。

MicrosoftCはQuickCとコンパチビリティを持っていますから、本書はQuickCユーザーの参考にもなるものと思います。しかし、特にQuickC使用時の注意は記述していません。

本書は「使いこなすためのポイント」に重点をおいて書かれています。そのため、実使用時には、もっと基本的な事柄、MS-DOSやマシンに関する資料も必須となります。これは各自で用意することをお勧めします。巻末に収録した参考資料一覧は、持っていて決して損のない書籍類ですので参考にしてください。

1989年2月　　菱木研二
斉藤　直

目次●*CONTENTS*

第3章 プログラミングのかんどころ

【リファレンス】

■ストリーム入出力関数

■コンソール入出力関数

■低レベル入出力関数

■ファイル操作関数

■システムコール関数

■プロセス制御関数

I

MicrosoftC Ver5.1の基礎知識

The Basic Knowledge of MS-C Ver 5.1

第1章　C言語とMS-DOS

前書きにも書いたとおり、この第1章ではMicrosoftCを使用するにあたって、最低限知っておかなければならないことを示します。

C言語の本を端から端まで読んでみたが、何のことかよくわからなかったなどという話をよく聞きます。C言語はマシンに依存した関数を持たないのが一般的です。したがって、その走る環境（マシンやオペレーティングシステム）についての知識を持ってこそ、初めてC言語を活用できるのです。また、C言語の中にも、いろいろな流行があります。たとえば、環境版のコンパイラなどがあげられます。そういった、MicrosoftCの実使用に密着した、最低限の知識をここでは紹介します。ただし、あくまでも最低限の知識ですから、その詳しい内容については、参考文献などを御参照下さい。

1.1 C言語について

ここ数年、C言語が脚光を浴びています。書店に行っても、かなりの種類のC言語の入門書が並んでいます。ちょっと前にはワープロの入門書が、その前はBASICの入門書が、同様にたくさん並んでいたものでした。

これはコンピュータ界のひとつのブームであることは否定できません。しかし、ブームになるにはそれなりの理由があるはずです。今までに無かった何かがありそうではありませんか。

この章ではC言語の背景と特徴、そして新しい流れについてのお話をします。すぐにC言語を使いたい人も是非読んでください。背景を追ってこそ初めてわかる使い方もあるのですから。

1.1.1 コンピュータとオペレーティングシステム

C言語は、UNIXのシステムを記述した言語として有名です。しかし当然、ある日突然でき上ったものではありません。では、一体どういう経緯を経て、今のC言語があるのでしょうか。

その話をする前に、ごく基本的なことをはっきりさせましょう。それは、コンピュータはいかにして動いているのかということです。

今日のようにパソコンが一般化し、日常品の中にまでコンピュータ制御の機器が氾濫している時代に、動作原理などどうでもよいと思われるかも知れません。あるいは、そんなことは知っているよと思われるかも知れません。しかし、あなたは今からC言語を使ってプログラムを書こうとしているのですから、ごく大雑把であっても把握して、あるいは復習しても損はないでしょう。

コンピュータはシリコンチップのかたまりです。そして、その中身はひとつひとつが重要なものの集合体です。その中でも重要なのがCPU(Central Processing Unit)であり、いわばコンピュータの中にいる指揮者です。ほとんどすべての処理はこのCPUを通して行なわれます。情報は一時的にRAM(Random Access Memory)に保存されます。漢字のフォントのように常に記憶しておくべきものはROM(Read Only Memory)に記憶されています。そして、外部とのコミュニケーションをはかるものとして、キーボードやCRT、フロッピードライブ、その他のポートがあります。

コンピュータの中では、すべての情報はONとOFFの組み合わせで表わされます。このONかOFFかという情報をビットといいます。数学的にいうならば、1と0の組み合わせの2進数ということになります。とはいっても、いかにコンピュータといえども設計したのは人間ですから、2進数ですべてのロジックが考えられるわけではありません。ビットを8つ組み合わせて8ビットを単位として色々な情報を考えています。この単位を1バイトといいます。1バイトを使えば2の8乗、つまり256の組み合わせが可能となります。

これがコンピュータ上での基本単位です。しかし、0と1の続く2進数8桁というのも読みにくいので、通常は0からfまでを使った16進で考えます。つまり、16進数2桁で16×16=256種類になるという考え方です。

処理できるのが16ビットであるという意味になります。当然のことながら、8ビットのコンピュータに比べて一度に処理できる量も、情報をためておくRAMの大きさも16ビットの方が多くなるわけです。

コンピュータの心臓部であるCPUには、そのCPU独特の命令があります。もちろんその命令もONとOFFの組み合わせが最も基本ではありますが、そのままでは到底人間にはプログラムが組めません。したがって数バイトでひとつの命令を形成してCPUに指示を送るわけですが、これとて16進数の組み合わせであり、とても理解しにくいものです。そこで、もう少し人間の言葉に近い形で命令できるようにしたのがアセンブリ言語と呼ばれるものです。

しかしながら、アセンブリ言語はCPUの命令と1対1です。これでは、複雑な処理をしようとする場合には大変面倒です。そこで、さらに人間の普段使っている言語に近づいたものが考えられました。これがBASICやFORTRAN、Pascalといった高級言語と呼ばれるものです。もちろんC言語も高級言語です。つまり、人間の言語により近いものを高級言語と呼んでいるわけです。

ここまでの話でおわかりでしょうが、コンピュータにおける言語は段々とユーザーフレンドリーに変貌してきています。この流れは現在も続いており、究極的にはコンピュータのプログラミングについての知識が無くてもプログラミングできるもの・・・すなわち人工知能へと向かっているわけです。

コンピュータはそのままではただの箱です。ソフトウェアがなければ何の役にも立たないがらくたです。本来、コンピュータにはオペレーティングシステム(OS)とよばれる極く基本的なプログラムがあります。オペレーティングシステムは、最低限の操作をするためのソフトウェアです。その上でワープロや、データベース、言語などのアプリケーションソフトを動かすのが本来の姿です。ちょっと前まであった、電源を入れるとBASICが立ち上がるなどといったパソコンは邪道であったと認識してください。

アプリケーション　ワープロ，データベースなど，いわゆるソフト
OS　マシンを動かす基本的なソフト
(コンピュータ)　マシン

オペレーティングシステムがあって、アプリケーションソフトがあって、初めてコンピュータは有効に使うことができるのです。

これは大変重要なことです。コンピュータ自体が違っても、共通のオペレーティングシステムならば、そのソフトウェアを走らせることができます。ただし、そのコンピュータのハードウェアに依存したルーチンを使っていなければの話です。たとえば、PC-9801と同じDOSであるMS-DOSを使っているIBM-PC/AT用MicrosoftCのライブラリは、PC-9801でもその大半を使用することができます。

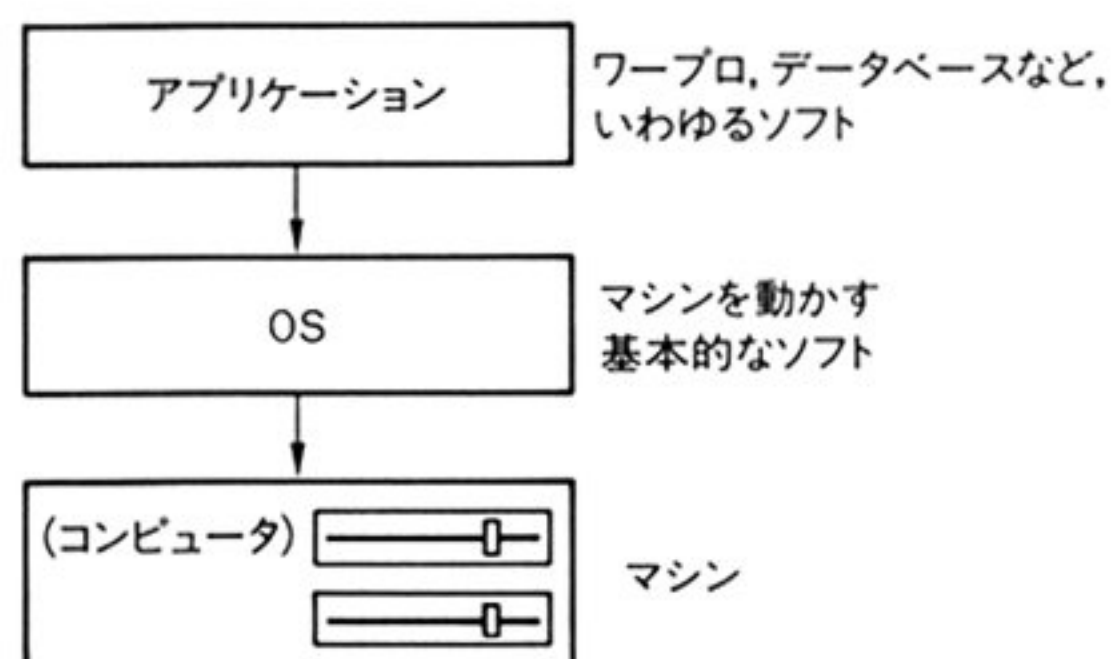
ところが、オペレーティングシステムが違えば、同じPC-9801でもCP/M用のソフトをMS-DOS上では動作させることができません。

C言語にはシステムに依存した関数についての、特定の取り決めがありません。したがって、ソースレベルではある程度の互換があり、ソースを持っていけば、他のオペレーティングシステム上のCコンパイラでコンパイルして動かすことが可能です。

1.1.2 C言語の歴史

C言語はAlgol60という言語の流れをくんでいます。これは、近代的な制御構造やデータ構造を持つもので、PascalなどもAlgol60の流れをくむものです。Algol60は名前が示すとおり1960年、国際委員会の手によって設計されました。

Algol60は構造化言語と呼ばれています。構造化とは、プログラムの構造を明確にして、読みやすく、そして書きやすくすることを主眼としたものです。具体的には、構文の規則性やモジュラ構造がはっきりしているということにあります。

しかし、Algol60はあまりに汎用なものを狙って開発されたため、かえって一般的には受け入れられませんでした。

そんなAlgol60を改良したのが1963年にケンブリッジ大学・ロンドン大学で開発されたCPL(Combined Programming Language)です。しかし、これも難し過ぎて一般には受け入れられなかったのです。

CPLの良い部分だけを抜き出したのが1967年にケンブリッジ大学で開発されたBCPL(Basic Combined Programming Language)です。このあたりから、ずいぶんと現在のC言語に近くなってきています。

そして1970年、アメリカのベル研究所でミニコンPDP-7の最初のUNIX用にBが開発されました。しかしBCPLやBはデータの型が少なくて応用範囲が限られていました。BCPLにいたっては機械語の型のデータしか扱えないものでした。

これらの問題を解決して汎用性を持たせたのが、1972年に同じくベル研究所でDennis Ritchieによって開発された「C」です。

C言語は複数の人間によるプロジェクトで作られた言語ではありません。そのため、ひとつの哲学とも言えるような一貫性を持っています。そして、C言語の持つ構造化プログラミングの書式も、製作者であるRitchieの著作により出版されています。日本では「プログラミング言語C」(B.W.カーニハン／D.M.リッチー著・石田晴久訳・共立出版)として発売されており、C言語を好むプログラマの間では、「K&R」または「バイブル」と呼ばれており、宗教的色彩すら帯びています。

実際、K&Rの書法は制御構造が書きやすく、かつ見やすいものであり、論理的なものです。これからC言語を始める人も、すでに使っている人にも是非読んでもらいたい1冊です。

近年、C言語が多くのプログラマに支持されている背景にはコードが小さいとか、速いとかいう以前にプログラマを魅了するような数学的な美しさがあるからではないでしょうか。

1.1.3 低級言語・高級アセンブリ言語としてのC

10,000行以上もあるといわれるUNIXのシステムプログラムの大部分はC言語で書かれています。先程のオペレーティングシステムを思い出してください。オペレーティングシステムはコンピュータの基本的なシステムですから、高速で、あまり大きくないコードであるべきです。したがって、オペレーティングシステムはアセンブリ言語を使ってその要求を満たすのが常です。それをC言語で書いてあるということは、それほどC言語によるプログラムがコンパクトで高速であるということです。

C言語の命令のひとつひとつは大変に簡単なものです。簡単というのは、使う人間にとってではなくコンピュータの機械語の命令に近いということです。さまざまな言語のうち、一般に人間の言葉に近い言語ほど高級言語とか高水準言語といいます。ですからC言語は、高級言語の中ではアセンブリ言語に近い低水準な言語ということができます。ある意味では、高級アセンブリ言語ということもできるかもしれません。

(人間)
高級言語(BASIC等)
C言語
アセンブリ言語
機械語
コンピュータ

最もアセンブリ言語に近いといえる特徴は、アドレス操作を持っているということでしょう。多くの高級言語は変数として数値型と文字型のような実際のメモリ上でのことを考えない型しか持っていません。ところが、C言語ではある変数のポインタ、すなわちメモリ上のアドレスを知ることができます。これは、システムに依存したプログラムを書く時には大変便利なものですが、ひとつ間違えると暴走を招く危険なものでもあります。これがまた間違えやすいものですから、注意してください。ポインタの扱いがわかればC言語の半分は理解できたと思ってもよいかも知れません。

また、C言語では入出力を言語仕様に含みません。つまり、ハードウェアやオペレーティングシステムに依存する部分は言語としては厳格な規定が無いの

です。したがって、システムに依存する部分は、コンパイラ側のライブラリに用意されており、ほとんどの場合ソースレベルでの修正をほどこさなくても、同じソースで他のオペレーティングシステム動作が可能なはずなのです。便利なユーティリティを作った時に、まったく別のオペレーティングシステムでも動作可能というのは、資源の有効活用という意味でもすばらしいものです。

また、論理型を持たない、実行時に型検査をしない、キャスト（強制的な型変換）ができる、ビットの操作が可能である、なとどというアセンブリ言語に大変近い特徴を持っています。

これらの特徴は、アセンブリ言語に近く、良質でスピードの速い実行ファイルができるということにもなりますが、その反面コンピュータの中でデータがどう扱われているかを理解していないと危険なプログラムを書いてしまう可能性も大いにあります。

また、コンパイラに最初から付いて来るライブラリの中には、でき上ると異常にファイルが大きくなるものがあり、実行時間の点からも不利になることもあります。

したがって、本当はあなたの使っているコンピュータのアセンブリ言語を理解した上でCを勉強すると理解しやすいはずです。

1.1.4 移植性

C言語にはシステムに依存した関数についての、特定の取り決めは無いことはすでに書きました。一般的にはこれがC言語の利点であり、ソースを持っていけば、他のオペレーティングシステム上のCコンパイラでコンパイルして動かすことが可能であると言われています。しかし、実際に役に立つようなレベルのプログラムを作った場合、他のCコンパイラやオペレーティングシステムへの移植はかなりの労力を必要とします。そのままコンパイルできて、期待通りの動きをすることはほとんど無いといってよいでしょう。

これには、いくつかの原因があります。これらの原因はそのままコンパイラの動作環境の差と考えることができます。低水準なところ、つまり人間と離れたところから順に考えてみましょう。その差を順に追っていくと、CPUの違い、ハードウェアの違い、オペレーティングシステムの違い、コンパイラの違いなどがあります。

CPUが違うということは、C言語の場合結構大きな問題です。C言語が元々動いていたUNIXは、中型コンピュータをベースに考えられています。これに対して、MicrosoftCの走るMS-DOSはインテル社製のコンピュータである8086系のものです。

8086系のCPUがセグメントとオフセットという16ビットずつのふたつのアドレスを使ってデータにアクセスするのに対して、モトローラ社の68000系CPUではひとつのアドレスしか持っていません。たとえば、8086系では団地の部屋を表わすために、「12棟の」＋「1325号室」と表わすのに対して、68000系では「121325号室」と表わすと思ってください。「121325号室」はそれだけでどの棟のどの部屋かがわかりますが「1325号室」だけではどの建物かわかりません。しかし、同じ棟の話しかしないということがわかっていれば、「1325号室」だけで話が通じます。

実際には8086系のCPUでは一度に連続して表わすことができるのは64Kバイトまでです。このために、色々な制限ができてしまうのです。コンパクトなプログラムで、一度に使えるデータ量も多くない時には、8086系が便利ですが、大きなデータなどを扱う場合は68000系の方が便利です。このため、64Kバイトを超えるデータを扱うための配慮がMicrosoftCにはあるのです。ところが、68000系のCPUではその配慮はまったく必要無いものとなってしまうわけです。

また、一度にCPUが計算できる数が違えば、同じ整数型でも精度が違ってきます。一般に16ビットコンピュータのC言語では、整数型(int)は16ビットですが、32ビットコンピュータでは32ビットです。したがって、変数の型自体が違ってくる可能性もあるわけです。

同じCPUでも、ハードウェアが違う場合には問題が出ることもあります。VRAMなどのメモリの構成やROMの内容が違えば、それに依存したプログラムは動作しなくなります。たとえば、直接あるアドレス上のROMデータにアクセスしているプログラムは、ソースを他のマシンでコンパイルしても、そもそもそのアドレスに目的のデータが無いために期待通りの動作はしないのです。

UNIXやXENIXなどのオペレーティングシステムでは、マルチユーザー・マルチタスクが実現されていますが、現在のMS-DOSはシングルタスクで

す。このために、ファイルに対する扱いがかなり変わって来ます。MS-DOSのファンクションコールなども、他のオペレーティングシステムでは実現されていません。

また、同じCコンパイラでも、関数が違えば、多少は言語仕様や関数も変わって来ます。同じ関数名でも引数の順番が違うこともありますから注意しなくてはなりません。

前に書いたように、アセンブラレベルのことや、マシンのハードウェアに密着したような、効率の良いプログラムにすればするほど、一般には移植性は低下してしまいます。もちろん、プリプロセッサ擬似命令を使って色々なシステムに対応することも可能ではあります。しかし、ハードウェアに密着したプログラムの場合は、実行効率・コード効率と移植性のどちらかを選択することが必要にならざるをえないでしょう。

1.1.5 開発性

C言語のプログラムは関数の集合です。ここからがプログラムの開始である、という合図のmainですら関数です。関数から、関数を次々に呼び出して、処理を行なっていくのがC言語でのプログラミングの基本型です。

mainではあまり細かいことをしないで、それぞれの仕事をする関数を呼び出すだけにします。それから1レベルずつ下の関数を作っていけば、見通しの良いプログラムを書くことができます。

つまり、mainなど、上のレベルのプログラムでは、大体のやっている中身がわかるような名前の、下のレベルの関数を呼び出します。下のレベルの関数では、さらに細かい処理を行ないます。それぞれの関数は階層状になっており、さらに下の関数へと続いています。そこだけ見れば、その部分についての大まかなアルゴリズムがわかるようにするのです。

このような書き方をすれば、全体的に見通しの良いプログラムを書くことができます。これを、構造化プログラミングと呼んでいます。

構造化プログラミングを実践すれば、多人数でのプログラム開発も可能になってきます。自分は、自分の担当する関数部分だけを考えればよいので、開発は随分と楽になります。しかし、このやり方をする場合には、全体の仕様をきちんと決めておかないと、バグが出た時にどこが悪いのかわからなくなってしまいますから、注意しなくてはいけません。

C言語の変数は、変数の宣言のしかたによって、寿命と可視性が変わってきます。

MicrosoftCでは、関数の外に取ってある変数、すなわち外部変数ではグローバルな寿命を持ちます。グローバルとは、そのプログラムのどこででも参照できるということです。

それに対して、関数内部で宣言されたもの、すなわちローカルな変数は他の関数内部で宣言された同じ変数名のものの影響を受けません。つまり、その関数内でのみ有効な変数なのです。したがって、一般的にはもう一度その関数を呼び出した時に、以前の値を保持しているとは限らないのです。もちろん、static宣言をすれば、以前の値を保持しながらも、他の関数に影響を与えないということも可能です。

こういった変数の特徴は、小さな関数を集めてひとつのプログラムを作る場合に大変有効なものとなります。多人数でひとつのプログラムを開発する場合に、他の人がどのような名前の変数を使ったかなどということは意識しなくて済むからです。

自分が作った低レベルの関数をたくさん持っていれば、新しいプログラムを作る時にも全部を作り直す必要はなくなります。以前に作った関数を呼び出せばよいのですから。したがって、後々に使うことを意識して書いた関数であれば、プログラムを作れば作るほど、後になるほど楽になるわけです。

もちろん、低レベルの関数は最初からそのコンパイラのライブラリとして付いてきますから、最初から全部自分で作る必要はありません。

1.1.6 C言語の新しい流れ

C言語の特徴を述べてきましたが、C言語はうまく使えば実にエレガントな言語です。しかしながら、マニアックな面も否定できません。BASICしか使ったことがない人が、C言語を使って最初からエレガントなプログラムを書けるかといったら、残念ながら難しいでしょう。それ以前に、BASICのようなインタープリタではなく、コンパイラであるC言語では、期待通りの動作をするプログラムを書くことすら難しいでしょう。

関数単位で書いて行くC言語では、括弧をひとつ書き忘れただけで、コンパイル時に鬼のような数の

エラーが出ます。本当にエラーなのは最初の1つ目だけであっても、括弧が無かったためにそのあとも随所がエラーとなってしまうからです。その事実を知らない初心者だったら、エラーの数だけでデバッグする気力もなくなるでしょう。

しかも、何行目がエラーだと出てから、再びエディタを立ち上げて、修正の後にセーブして、再びコンパイル、リンクしなくてはいけません。たとえ、コンパイラを通ったからといって、ちゃんと動くという保証はありません。実行結果が期待通りでなかったら、再びエディタを立ち上げて･･･ということの繰り返しです。

これは、慣れてしまえばなんということはないのですが、BASICなどのインタープリタに慣れている人にとっては、苦痛以外の何物でもないでしょう。

たとえば、ある変数の内容の数字が期待通りに表示されないとしましょう。プログラム中のあるところでの、その変数の内容を調べたいとき、BASICならばそこで止めてから「print a」などという荒技も可能です。しかし、C言語の場合は止めたらそこで終わりですから、ソースを書き直してprintf()をはさみ、再度コンパイルしなくてはいけません。これは慣れていない人にとっては時間を食うばかりです。

また、C言語ではBASICには装備されていたグラフィックやFM音源などに関する関数は今までは付いてきませんでした。使用するためには、マシンの解析資料などを元に自分で関数を作っていたわけです。これも初心者にとっては辛いものです。C言語なら高速にグラフィックも使えるだろうと思っても、自分で関数を作らなければいけないのですから･･･

このようなC言語に対する取っ付きにくさをなんとかしようという流れが最近出てきました。そのひとつが統合開発環境です。

統合開発環境とは、今までバラバラだったエディタ、コンパイラ、リンカ、デバッガをひとつにまとめてしまったものです。つまり、毎回これらのひとつひとつを立ち上げ直さなくても、デバッグを可能にしたものです。しかも、ソースをメモリ上に持っていますからコンパイルの速度も速くなっています。

現在、統合開発環境を実現したものには、マイクロソフト社のQuickCとボーランド社のTurbo Cがあります。QuickCはTurbo Cに比べるとデバッグ機能が重視されています。

Turbo Cでは、統合型のtcとコマンドライン版のtccでは、基本的に同じものが同じようにコンパイルされるのに対して、QuickCとMicrosoftCでは若干の違いがあるので注意が必要です。

機種依存しているので、他のマシンでは使用不能

また、MicrosoftCのバージョン5.1からはグラフィックや、MS-DOSのファンクションコールの関数が標準で付いてきます。Turbo Cではバージョン1.5・PC-9801版からは、それらに加えてFM音源の関数も付いています。

これらの関数が付いてくることは、プログラマの負担が減るという意味では大変良いことなのです。しかし一方では、MS-DOSならどのマシンでも走っていたコンパイラが、完全に機種依存になってしまい、他のマシンでは動かなくなってしまいました。Turbo Cでは、テキストVRAMに直接書き込む関数まで付いてきています。また、Turbo Cのバージョン2.0では、統合環境版だけでなく、コマンドライン版のコンパイラ自体が機種依存になっています。

高速になることも、高機能になることも良いことですが、一方ででき上ったプログラムのサイズは大きくなるばかりです。統合開発環境ではたくさんのメモリを必要とするために、一度に大きなプログラムはコンパイルできないなどということも出てきています。

最近のソフトウェアは、何につけメモリを無駄遣いしているように感じられます。こういった流れは、一方ではコンパクトで高速でエレガントというC言語本来の姿とは相反するものであることも否定できません。

1.2 MS-DOSとC言語

1. 1. 4で述べたとおり、同じC言語でもオペレーティングシステムによって、色々な制限や違いがあります。MicrosoftCを使いこなすためには、MicrosoftCが走っているオペレーティングシステムであるMS-DOS、そしてMS-DOSが走っているCPUである8086系CPUについての理解が必要不可欠です。

この節では、MicrosoftCの環境であることを前提にMS-DOSを紹介します。

1.2.1 8086とセグメント

MS-DOSは8086, 8088, 80286, 80386などのCPUのためのオペレーティングシステムです。MS-DOSについてお話する前に、この8086などのCPUについて紹介しましょう。

これら8086などのCPUは、Intel社によって設計され、10社ほどのメーカーがセカンドソース品を製造しています。これら8086系のCPUは世界で最も多く使われている16ビットCPUのひとつです。

8086は、同社の8080（Z80は8080の上位コンパチCPU）という8ビットCPUに比べて、処理能力が10倍、メモリ・アドレス領域が16倍になっています。

8088は8086に近いスペックを持つ8ビットCPUで、レジスタは16ビットになっているものです。

80286は、8086とソフトウェアで互換性を持ちながら、6倍のパフォーマンスとメモリマネージメントと、メモリプロテクション機能を持ったもので、論理アドレス空間は1Gバイト、物理アドレス空間は16Mバイトを持っています。

80386は、80286とソフトウェアで上位互換性を持つ32ビットCPUです。ページング方式の仮想記憶をサポートし、論理アドレスは64テラバイト、物理アドレス空間は4Gバイトを持っています。

8086は、高速・高精度の浮動小数点演算プロセッサである8087や、高度なI/Oコントロールを可能にした8089などを追加することによって、より高度で広範なアプリケーションに対応できるようになっています。

8086などの名称は旧称であり、現在の正式名称は、

iAPX86/10(8086)
iAPX88/10(8088)
iAPX86/20(8086と8087のシステム)
iAPX88/20(8088と8087のシステム)
iAPX286/10(80286)

などとなっています。

PC-9801では、PC-9801/E/F/Mには8086が、PC-9801VM/VF/U/UVには8086のアッパーコンパチであるV30が、PC-9801VX/UX/RXには80286が、PC-9801RAには80386が、それぞれ搭載されています。

8086では、1Mバイトまでのアドレスを扱うことができます。1Mバイトのアドレスを扱うためには、20ビット（16進数としては5桁）の情報が必要になります。8086では、16ビット（16進数としては4桁）のレジスタを2つ使って、20ビットのアドレスを表わしています。上位・下位それぞれのアドレスをセグメントアドレス・オフセットアドレスと呼んでいます。実際には、4ビットシフトしたアドレスの加算によって実アドレスを得ています。

たとえば、セグメントアドレスが9876H、オフセットアドレスが8765Hであれば、実アドレスは、

```
     9876   セグメント 16ビット
 +    8765  オフセット 16ビット
 ─────────────────────
    a0ec5   実アドレス 20ビット
```

という具合に計算されます。

ここで、疑問に思いませんか? 最初から20ビット分を一度に計算するようにできていれば、こんな面倒な計算をしなくてもいいはずです。なぜ、こんなまわりくどいことをしているのでしょう。

16ビットで表わすことのできる最大の数は64Kバイトです。つまり、4桁の16進数では64Kバイトまでのデータを扱うことができるわけです。64Kバイトというのは、8ビットのCPUで扱えるアドレス空間と同じです。

ちょっとしたプログラムは、64Kバイトなどという大きさにはなりません。小さなプログラムや、少ないデータならば、64Kバイトあれば十分扱うことができます。それならば、最初にセグメントだけを宣言しておけば、あとはオフセットのアドレスだけでプログラムはデータにアクセスできます。したがってプログラムを小さなコードで書くことができる

わけです。

先ほどの例で考えてみましょう。a0ec5Hというアドレスを表現するとき、5桁が必要になりますが、8086では最初にセグメントが9876Hであると宣言しておけば、4桁の8765Hだけでアクセスできます。ここで1桁違っても大したことはないように思われますが、機械語レベルで考えたとき、メモリへのアクセスは、すべて1バイト少なくなってしまいますから、実際にはかなりプログラムは小さくなるわけです。

これは、8086系CPUの大きな美点であると共に、大きな落し穴でもあります。つまり、64Kバイトを超えるような連続したデータを扱おうとすると、非常に複雑で面倒な作業が必要になるわけです。

MicrosoftCでは、データやプログラムの大きさが64Kバイト以下であるか、越えるかによって、作り出されるコードが変えられるように設計されています。すなわち、64Kバイト以下であれば、コンパクトなコードのプログラムが作成されます。64Kバイトを超える時は、コードは大きくなり、メモリーのアクセスも複雑になるので、実行速度も遅くなります。

具体的には、64Kバイト超える(○)、超えない(×)によって、次のような分類がされています。

メモリモデル	プログラム	データ
スモールモデル	×	×
ミディアムモデル	○	×
コンパクトモデル	×	○
ラージモデル	○	○
ヒュージモデル	○	○

MicrosoftCでプログラムを作成する時には、プログラムの大きさ、扱うデータの大きさによって、これらのメモリーモデルを指定しないといけません。メモリーモデルの指定は、コンパイル時にオプションとして指定します。指定がない時は、スモールモデルとしてコンパイルされます。このため、ライブラリも各モデルに対応したものが用意されています。このような特徴を踏まえた上で、プログラムを作らないと、思ったように動作しないということが起きる可能性があるわけです。

8086のセグメントを管理するレジスタは4本あり、それぞれ、

CS(コード・セグメント)
DS(データ・セグメント)
SS(スタック・セグメント)
ES(エクストラ・セグメント)

と呼ばれています。

CSは、CPUが実行するプログラムのコードが格納されているセグメントです。DSおよびESはデータ転送時に使用されます。SSはスタックに使われます。

オフセット・アドレスの指定には、次の5つが扱われます。

IP(インストラクション・ポインタ)
SP(スタック・ポインタ)
BP(ベース・ポインタ)
SI(ソース・インデックス)
DI(デスティネーション・インデックス)

CSに対応するものはIPです。

レジスタにはこれらの他に、主に演算などに使用されるものとして

AX(アキュムレータ・レジスタ)
BX(ベース・レジスタ)
CX(カウンタ・レジスタ)
DX(データ・レジスタ)

があります。これらのレジスタは16ビットレジスタですが、8ビットを単位としても扱うことが可能です。AXでしたら、AHとALのように、highとlowのふたつで表わされます。

C言語のプログラムは一般に小さいと言われますが、アセンブラで作ったプログラムに比べれば、随分と大きくなります。そのため、MicrosoftCは、コンパイル時にオプティマイズを行ないます。オプティマイズとは最適化と呼ばれ、実行速度優先・サイズ優先などの指定が可能です。しかし、あまりオプティマイズを強力にし過ぎると、時として予想もつかないコードを作り出すことがあります。こんな時や、どう直しても思い通りにプログラムが動作しない時、バグに悩まされた時などは、アセンブラ出力をしてみるのもひとつの手です。

CodeViewなどでのデバッグの際も、CPUに関する知識はあるにこしたことはありません。MicrosoftCを使いこなすためには、8086のアセンブラも勉強しておくことをお勧めします。

1.2.2 MS-DOS

前にお話したとおり、オペレーティングシステム

は必要最低限の入出力をするためのプログラムです。エディタやワープロ、コンパイラ、リンカなどは、すべてオペレーティングシステム上で動いています。

MS-DOSもオペレーティングシステムです。したがって、MS-DOSがコンパイルするわけでも、MS-DOSがワープロになるわけでもありません。MS-DOSは、あくまで最低限のことをするオペレーティングシステムでしかないのです。ですから、ほんの基本的なコマンドさえ知っていれば、MicrosoftCを使うことはできます。MicrosoftCがセットアップしてあれば、

＞cl test.c

と打てば、コンパイルとリンクすることでしょう。

しかし、本格的なデバッグをするためには、MS-DOS上で動く、各種ツールの使用法をマスターしなければいけません。そして、そのツールを使い込むにつけ、マシン語レベル、MS-DOSレベルで、どのようにコンピュータが動作しているかを知る必要が出てくるでしょう。したがって、MicrosoftCを使い込もうとするならば、MS-DOSの詳しい解説書や、PC-9801のテクニカルデータブックなどを常備することをお勧めします。これらの本は、読み物としてではなく、あくまでデータとして、プログラムの開発には必需品です。

ここでは、詳しいデータではなく、知識としてのMS-DOSについて、お話しましょう。

PC-9801のMS-DOSのメモリ配置は、図1.1のようになっています。

図1.1　【MS-DOSのメモリ配置(バージョン3.1)】

アドレス	内容
00000H～	割りこみベクタ
00400H～	MSDOS版BASICワーク用エリア
00500H～	BIOSワークエリア
00600H～	IO.SYS
068D0H～	MSDOS.SYS
0D2F0H～	DISK BUFFER,DEVICE DRIVE BUFFER
	COMMAND.COM 常駐部
	TPA
	COMMAND.COM 非常駐部
A0000H～	VRAM
	未使用
E8000H～FFFFFH	ROM

00000Hからは、割り込みベクタ領域となっています。

MS-DOSでは、割り込みという手法を使って処理を行なっています。割り込みには、キーボードから入力などのハードウェア割り込み、MS-DOS内部のルーチンによって時間の演算を行うなどソフトウェア割り込みなどがあります。

割り込みが発生すると、MS-DOSはメモリの00000Hから用意されている割り込みベクタテーブルを参照します。割り込みベクタテーブルには、その処理を行うルーチンの位置が格納されています。実際には、セグメント2バイトと、オフセット2バイト、計4バイトの組み合わせの256種類、定義が可能です。

割り込みは8086のアセンブラで言うところの「INT」命令を使用します。MS-DOSのファンクションコール（あらかじめ用意されているルーチン）の中でも、最も標準的なものに「INT 21H」があります。これは、21H番目のベクタを使用するという意味です。実際には、AHレジスタを始め、いくつかのレジスタにパラメータをセットして、「INT 21H」を行うとMS-DOSの機能を利用できるわけです。実行結果についても、レジスタに結果の内容が返ってきますので、簡単に知ることができます。

たとえば、AHレジスタに02H、DLに文字をセットして「INT 21H」を実行すれば、スクリーンに文字を出力することができるといった具合です。

MS-DOSファンクションコール以外にも、PC-9801のBIOSファンクションコールがあります。これにはいく種類か用意されており、CRT・キーボード関係は「INT 18H」、ディスクは「INT 1BH」、RS-232Cは「INT 19H」、プリンタは「INT 1AH」といった具合に決まっています。

MicrosoftCのライブラリだけですべてのことはできません。割り込みを使わないと実現できないことも数多くあるのです。また、割り込みを使うことによって、よりマシンに近いレベルでの、きめの細かいプログラムを書くことができます。

割り込みの制御はレジスタに各種パラメータをセットして行ないますから、アセンブラで書いた方が簡潔かつ、わかり易いプログラムが書けるでしょう。C言語でも割り込みは使うことができます。割り込みを使うとどうしても冗長になりがちではありますが、使う意義は大きいのですから、是非マスターしてほしいポイントです。

割り込みを使ったプログラムを記述するためには、そのためのパラメーター一覧が掲載してある書籍が必要です。「PC-9801シリーズ・テクニカルデータブック」(アスキー出版)、「MS-DOS Ver.3.1活用ハンドブック」(秀和出版) などが最適であると思われます。

00400Hからは、MS-DOS版BASICのワークエリアになっています。

00500Hからは、PC-9801のBIOSのワークエリアです。MS-DOSもこのBIOSを、ごく低レベルで使用しています。

00600Hからは、IO.SYSがロードされています。IO.SYSは、ハードウェアの制御をするためのルーチンの集まりです。IO.SYSの大きさは、バージョンによって違います。したがって、この後ろにあるMSDOS.SYSの先頭番地も、バージョンによって変わってきます。

MSDOS.SYSは、MS-DOSの中核であり、ファイル管理、メモリ管理、プロセス管理などをします。MSDOS.SYSも、バージョンによって大きさが違います。

この次には、デバイスドライバおよびMSDOS.SYSワークエリアが配置されています。デバイスドライバエリアには、ユーザーがconfig.sysで登録したデバイスドライバのルーチンが配置されています。MSDOS.SYSワークエリアは、実際にはconfig.sysで指定されたディスクバッファのエリアです。したがって、このエリアの内容はconfig.sysによって、大きく変わります。

COMMAND.COMは、コマンドインタープリタとも呼ばれ、ユーザーとMS-DOSとの接点となるものです。COMMAND.COMには大変多くの機能があるため、かなりのメモリを消費します。しかし、ユーザーのプログラムが動作中は、その大半は必要ありませんから、最低限必要な常駐部だけが存在しています。常駐部は破壊されることはありませんが、非常駐部はユーザーのプログラムによって破壊されることがあります。もっとも、非常駐部は破壊されても再びロードし直されますから問題ありません。

TPAには、ユーザーのプログラムがロードされます。この後ろにCOMMAND.COMの非常駐部がありますから、破壊しても問題はありません。

A0000H以降のエリアは、PC-9801シリーズではVRAMとROMとして使用されています。したがってPC-9801シリーズのMS-DOSは640Kバイトのフリーエリアになっています。このエリアを、直接操作するようなプログラムは、他のマシンでは正常に動作しません。

同じMS-DOSといっても、このようにマシンによってメモリの使われ方が違います。したがってMicrosoftCでも、グラフィックなどを操作する関数などはマシンに依存したものです。MicrosoftCでは、バージョン3.0までは日本語版とIBM版との違いは日本語処理関係だけでしたが、バージョン4.0からはCodeViewというデバッガが付属してくるようになりました。このCodeViewは、テキスト画面のダイレクトアクセスをしているために、PC-9801専用となっています。バージョン5.1からはグラフィック関係の関数もサポートされましたから、MicrosoftCもPC-9801専用となりました。

グラフィック以外にも、PC-9801でないと動かない条件はありますが、グラフィック画面を使用しているかどうかということは、他のマシンで動くかどうかのひとつの判断材料となります。

MS-DOSでプログラムを作るときには、使用する関数や、割り込みの命令によって、使えるマシンが制限されます。ターゲットのマシンを制限しないならば、それなりに考えてプログラムを作らなければいけません。このように、MS-DOSとひとことに言っても違いがありますから気を付けないといけません。

今までの話でおわかりと思いますが、MicrosoftCで本格的なプログラムを作成するためには、MS-DOSや8086についての知識が必要不可欠です。MS-DOSやPC-9801に関して詳解してある書籍などを手に入れ、資料として常備することをお勧めします。

1.3 MicrosoftC

この章では、今までBASICなど他の言語を使用してきたユーザーが、MicrosoftCを使うために必要な知識や常識といったことを紹介します。

また、MicrosoftCはどのようなものであるか、どのようなメリットがあるのかといったことを、他の言語との比較や、他のCコンパイラとの比較などを通じて、明確にしていきたいと思います。

1.3.1 C言語 vs BASIC

MicrosoftCの正式名称は、「Microsoft C 5.1 Optimizing Compiler」です。名称にもあるように、MicrosoftCはコンパイラです。「コンパイラだから、Cは速い」とか、「インタープリタだから、BASICは遅い」などとよく言われますが、コンパイラとは一体なんなのでしょう。そして、インタープリタとはどういうものなのでしょう。

ここではコンパイラであるC言語とインタープリタの代表であるBASICとを比較してお話しましょう。

1.3.1.1 コンパイラとインタープリタ

PC-9801には、最初からBASICのフロッピーディスクがついてきます。このフロッピーディスクを入れてPC-9801を立ちあげるとBASICが起動します。MS-DOSなどが無くてもBASICはいきなり使うことができます。もちろん、MS-DOS版のBASICもありますから、MS-DOSからBASICを起動することも可能ではあります。

MS-DOSで言うところの「DIR」にあたるような、ファイル一覧を表示する機能「FILES」なども、BASIC上でそのまま使えます。したがってBASICを一度立ちあげてしまえば、それだけで一通りの作業が可能です。そのままプログラムを書き始めて「RUN」と打てば実行します。間違いがあればそこで実行を停止し、エラーを表示します。BASICでは、プログラムを1行ずつ機械語に翻訳して実行します。したがって、実行が不能となったときには、そこで停止します。このように、プログラムの実行と同時にプログラミング言語から機械語に翻訳して実行するものをインタープリタと呼んでいます。

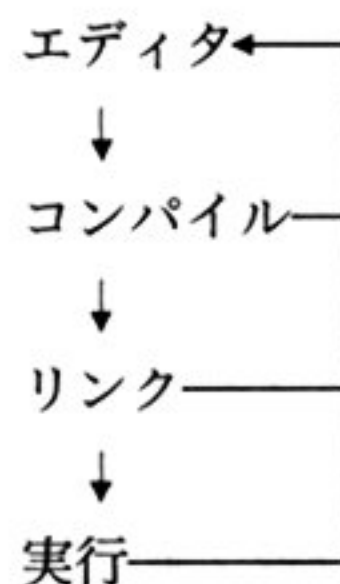
これに対して、プログラミング言語で書かれたものを、一度、機械語に直してしまってから、実行するものをコンパイラと呼んでいます。この機械語に直す作業をコンパイルと呼んでいます。

「コンパイラは速い」というのは、実行するときは完全に機械語になっているからです。コンパイラで作られた実行形式は機械語だけで構成されています。いちいち翻訳することがないので、実行時には速いプログラムとなるわけです。実際には、インタープリタに比べてコンパイラで作成されたプログラムは10倍以上も高速です。

実行速度は速いにこしたことはありません。では、インタープリタのメリットとはなんでしょう。これは、そのままインタープリタとコンパイラの環境の違いに集約されます。

コンパイラは、MS-DOS上でいろいろな作業が必要となります。

まず、エディタでその言語のソースプログラムを書きます。プログラムが一通り書けたならば、コンパイラでコンパイルします。エラーが出たならば、エディタで書き直します。これをエラーが出なくなるまで繰り返さないと、実行可能なプログラムはできません。コンパイルが成功したら、あらかじめコンパイラに用意されているライブラリとリンクします。リンクでエラーが起きれば、それに対処するために、再びエディタを起動してソースを書きかえるという作業が必要になることもあります。リンクが成功すると実行形式のファイルができますが、実行してみて期待通りでない場合は、またしても、エディタに戻ってソースを書きかえる作業が待っているわけです。

```
エディタ←─┐
  ↓        │
コンパイル─┤
  ↓        │
リンク─────┤
  ↓        │
実行───────┘
```

これに対して、インタープリタは先ほどお話したとおり、BASICという言語自体がひとつの環境とな

っています。つまり、エディタやMS-DOSなどを全然意識しなくても実行可能なプログラムが開発できるわけです。しかもBASICでは、エラーが起きたときや、実行中にプログラムを停止して変数の内容を表示しても、再度実行を開始したりすることもできます。

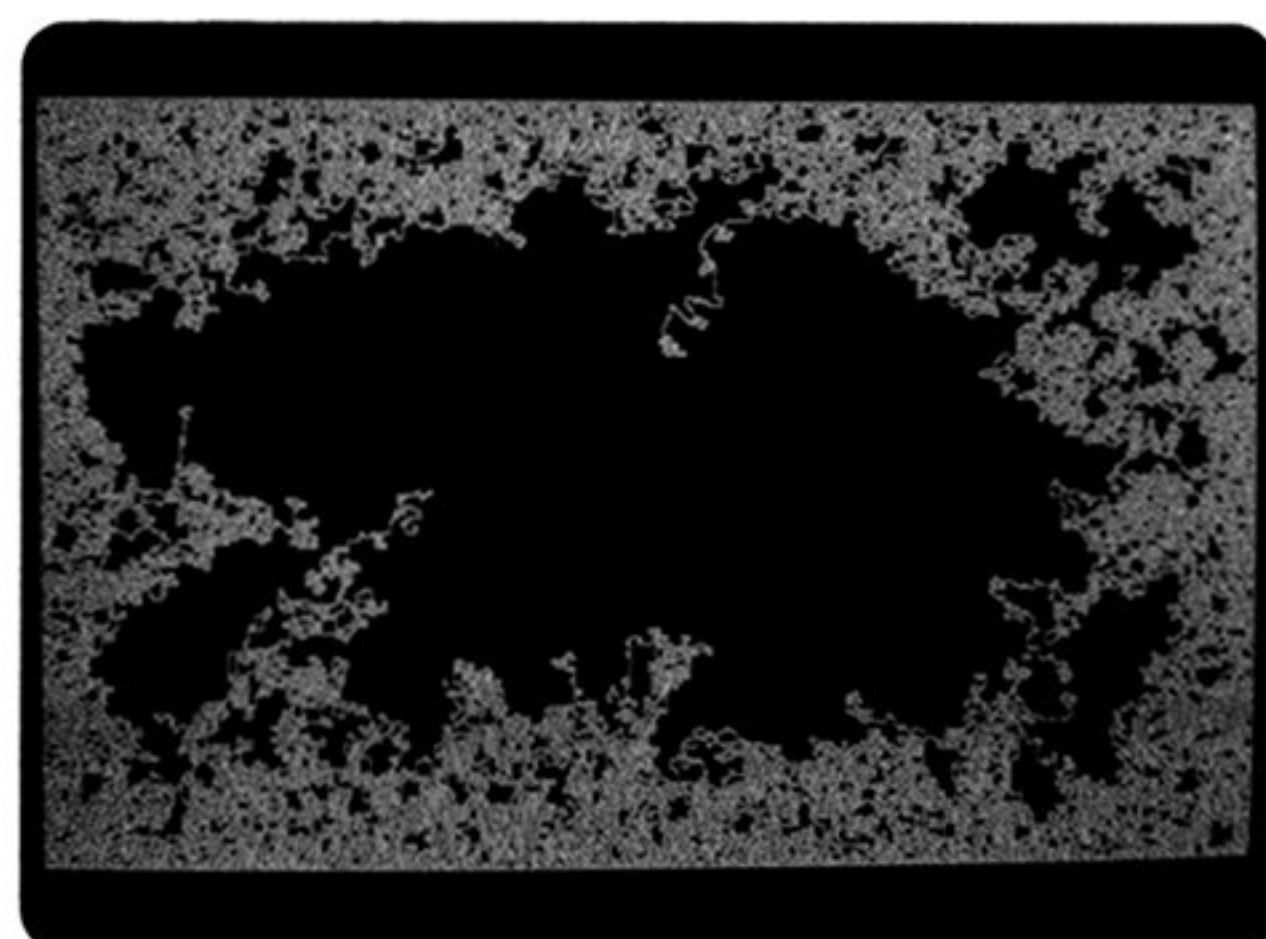
この比較を見る限り、コンパイラは面倒なだけでプログラムを開発する上では能率が悪いばかりのような印象すら受けます。しかし、実際にはこのようなことだけでは済まないのです。

たとえば、ファイルを作り出すプログラムを作ったとします。C言語で先ほどのような過程を経てプログラムを作成したのであれば、でき上ったファイルを「dump」コマンドで内容を確認することができます。ところがBASICでは、確認する手段がありません。内容をダンプするプログラムをあらかじめ作っておいて、今のプログラムをSAVEしてから、それを実行するか、MS-DOS版だったら、MS-DOSに戻って実行しなくてはなりません。MS-DOSではプログラム開発に便利な各種のツールが用意されているのに、BASICではそれらは利用不可能なのです。

また、MS-DOS版BASICを使って、MS-DOSを使用しているときに便利なプログラムを開発したとしましょう。ところが、インタープリタであるBASICでは、それを実行するためにはBASICを立ちあげてから実行しなくてはなりません。これは、いくらなんでも間抜けです。

ソースのエディット画面

MicrosoftCのコンパイル画面

MicrosoftCのリンク画面

プログラムの実行画面

BASICの画面

MS-DOSを前提にして、実際に役に立つようなプログラムを作ろうと考えてくると、BASICインタープリタでは事実上無理があります。そこで考えられたのが、BASICコンパイラです。

BASICコンパイラは、BASICインタープリタで作成され、実行が可能であるものをコンパイルして実行形式を作るものです。これならば、デバッグをBASICインタープリタで行なって、インタープリタより高速な実行形式を作成することはできます。

QuickBASICのエディタ画面

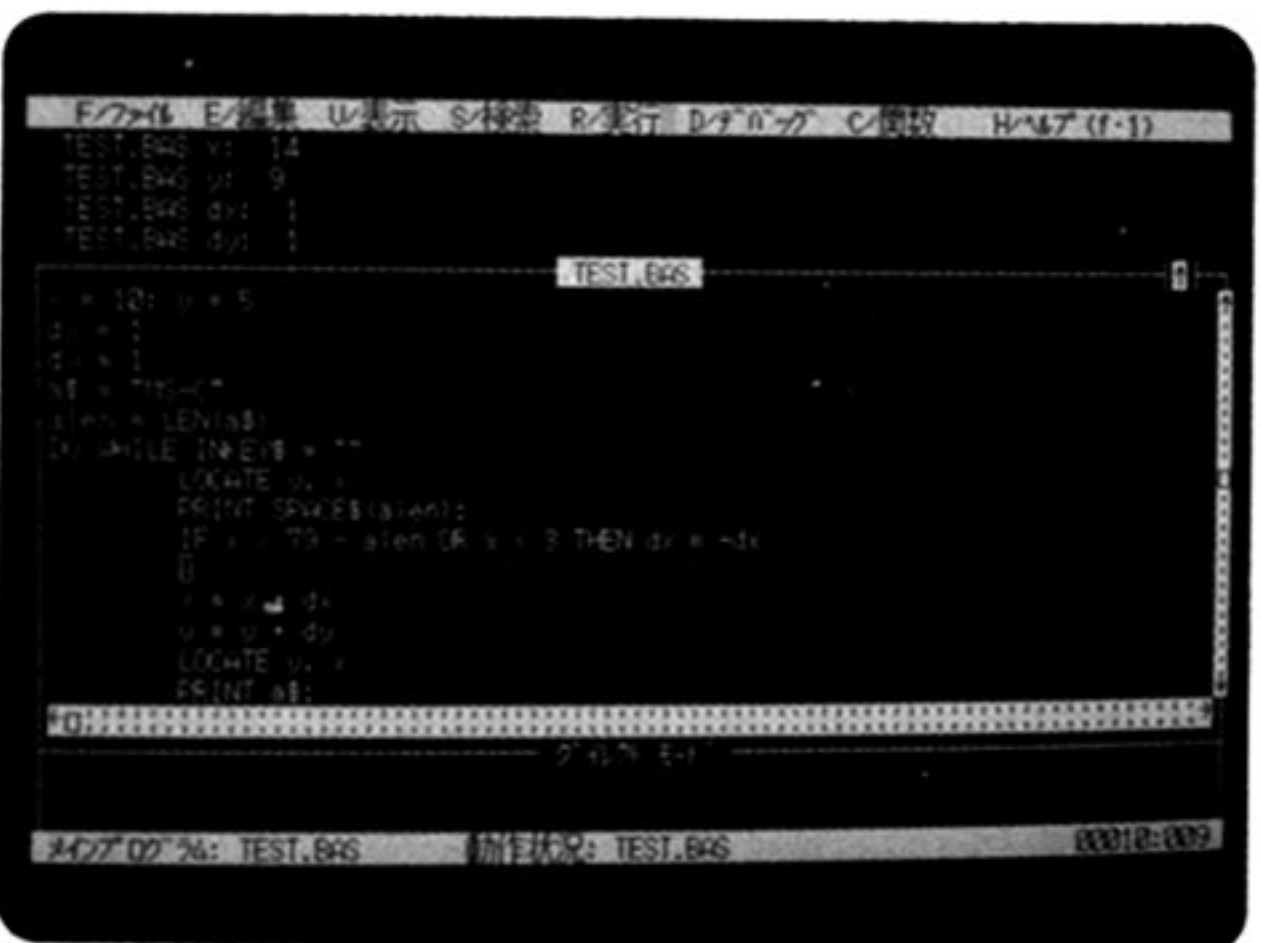
同、デバッグ画面

しかし、BASICコンパイラとはいっても、C言語などのコンパイラとは違って、いくつかの不都合があります。BASICコンパイラと一般に呼ばれているものは、完全な実行形式ファイルを作ることができません。BASICコンパイラでは、Pコードと呼ばれるコードに翻訳され、別ファイルとして存在しているランタイムライブラリを呼び出して実行します。したがって、C言語で作成されたものに比べると、比較にならないほど遅いものとなってしまいます。また、数百Kバイトもの恥ずかしいランタイムライブラリを必要とします。これは、いくらなんでも、エレガントとは言えません。

最近では、MicrosoftCのメーカーと同じMicrisoft社から、QuickBASICというBASICが発売されました。QuickBASICは、今までのBASICと同様にインタープリタとしてのBASIC、ランタイムライブラリを必要とするBASICとともに、純粋に実行形式も作成できます。

1.3.1.2 言語自体の違い

QuickBASICの登場により、同じコンパイラとしてのMicrosoftC vs BASICの対決が可能となりました。そこで、QuickBASICとMicrosoftCを使って、同じ動作をするプログラムを作成してみました。

内容は「MS-C」という文字列が、画面をボールのように跳ね回るというものです。画面の端に文字列がぶつかると跳ね返り、画面上の文字を消していきます。何かキーが入力されるまで、この動作は続きます。list.1にQuickBASICのソース、list.2にMicrosoftCのソースを示します。

同、コンパイル画面

```
------------------------------------------------------------------
list.1  bcls.bas

x = 10: y = 5
dx = 1
dy = 1
a$ = "MS-C"
alen = LEN(a$)
DO WHILE INKEY$ = ""
        LOCATE y, x
        PRINT SPACE$(alen);
        IF x > 79 - alen OR x < 3 THEN dx = -dx
        IF y > 23 OR y < 2 THEN dy = -dy
        x = x + dx
        y = y + dy
        LOCATE y, x
        PRINT a$;
        FOR i = 0 TO 150
        NEXT i
LOOP
------------------------------------------------------------------
list.2  ccls.c


#include    <stdio.h>
#include    <conio.h>
#include    <string.h>

locate(int y, int x)
{
        printf("¥x1b[%d;%dH", y, x);
}

main()
{
        int     x, y, dx, dy, i, alen;
        char    *name = "MS-C", *buf = "    ";

        alen = strlen(name);
        dx = 1;
        dy = 1;
        x = 10;
        y = 5;

        while (!kbhit()) {
                locate(y, x);
                printf(buf);
                if ((x > 79 - alen) || (x < 3))
                        dx = -dx;
                if ((y > 23) || (y < 2))
                        dy = -dy;
                x += dx;
                y += dy;
                locate(y, x);
                printf(name);
                for (i = 0; i < 150; i++)
                        ;
        }
}
------------------------------------------------------------------
```

それぞれのプログラムは、ソースレベルでのアルゴリズムはまったく同じです。ところが、実際にコンパイルしてみると、表1.1のように、でき上がったサイズに大きな違いが出ました。このプログラムで見る限り、サイズはQuickBASICの方が5倍も大きくなります。

表1.1

BCLS	BAS	357
BCLS	EXE	38347
CCLS	C	535
CCLS	EXE	7529

実行してみると、MicrosoftCで作成されたプログラムは、速くて何をやっているのかわからないほどでした。そこで、画面に「MS-C」という表示が見えるようにするためのウェイトを、150から2000に変更しました。それでもMicrosoftCの方が2倍近く速く感じられました。

BASICとはいえ、QuickBASICはちゃんとしたコンパイラです。いったいなぜ、このような違いが出てくるのでしょうか?

1．1．1で、C言語は、低級な高級言語であるという話をしました。C言語は人間にわかりやすいかたちの言語ではありますが、マシン語レベルに近いものであるということです。つまり、一見似ている命令であっても、MicrosoftCとQuickBASICでは内部処理されるときに違いがあります。その違いがコードの大きさや、実行速度に反映されているわけです。

たとえば、BASICでは、A$という文字列変数を使用するために、文字列のバッファを宣言する必要がありません。したがって、自由に文字列の長さを増やしていけます。しかし、C言語では最初に宣言したバッファを超えると他のデータを壊してしまうため、最悪の場合には暴走することもあります。もっとも、暴走しなくても、気が付かないうちにデータを壊していた場合の方が始末が悪いかもしれません。BASICではそのような心配をしなくても、言語側で処理してくれる代わりに速度とコードのサイズが犠牲になっているわけです。

BASICでは「LOCATE」という命令を何気なく使えます。LOCATEは、キャラクタ画面上のどこにカーソルをもっていくかという命令ですが、今までのC言語ではそのような命令は用意されていませんでした。こんな単純で基本的とも思える命令が用意されていないなどとは、BASICを今までやってきた人にとっては信じられないことかもしれませんが、そればかりか、グラフィックに関する関数すらなかったのです。

テキスト画面の制御やグラフィックに関する命令は、マシンによって制御方法が違うので、サポートしていませんでした。つまり、いままでのMicrosoftCはMS-DOSなら何でも動くコンパイラであり、付属のライブラリも特定の機種用のプログラムを作成することを前提に作られてはいなかったのです。マシンに依存したプログラムを作成するためには、プログラマがそれに対応した関数を作ることが必要だったのです。

C言語は関数の集合であるという話をすでに書きました。関数から関数を呼んで、プログラムを実行していきます。printfなど、画面に出力するような最も基本的とも思えるようなものも関数として「stdio.h」に宣言してあります。付属のライブラリを使用するときには、その関数が宣言されているインクルードファイルをインクルードしなくてはいけません。プログラム中で使用されている関数で、あらかじめMicrosoftCに付属のものは、その関数の下請け関数とともにリンク時にライブラリから抜き出されてリンクされ、実行形式が作り出されます。したがって、実行形式ではそのプログラムの実行に必要な最小限のルーチンだけの集合となるわけです。

もしも、必要な機能を満足している関数が無い場合は、自分で作らなければならず、かつてはグラフィックなどに関する関数も自作したものでしたが、MicrosoftCもバージョン5.1からは、グラフィックなどに関する関数が標準でサポートされています。実は「LOCATE」に対応するような「settextposition」という関数も用意されています。list.2では、「locate」という関数を自分で作っています。この関数は、エスケープシーケンスを利用して座標を指定するものですから、PC-9801でなくても、このエスケープシーケンスをサポートしているマシンなら動作します。これに対して、「settextposition」は、PC-9801のVRAMを直接操作しているようです。この関数は「graph.h」で宣言されていますから、「graph.h」をインクルードしなくてはいけません。ところが、この関数を使用すると、この関数の機能を実現するためのいろいろな下請けの関数がリンク時に自動的にくっ付いてきてしまいます。実際に「settextposition」を使って同様なプログラムを作成したところ、なんとサイズは50Kバイトを越えてしまいました。

それではQuickBASICでも、使わないライブラリがあれば切り捨てるのでしょうか。ためしに「i＝10」というだけのプログラムをコンパイルしてみました。すると、36Kバイトほどの大きさになっていました。どうやら、QuickBASICでは30Kバイトほどの

ランタイムライブラリはかならずくっ付いてくるようです。

MicrosoftCを使用する美点は、使用目的に合わせて関数を選べることです。たとえば、画面出力ひとつをとっても、printf、puts、putc、putchar、putw、cprintf、cputs、putchなどと、多種多様なものが用意されています。printfは、出力フォーマットを指定できる高機能なものですが、これを使うことによってでき上る実行形式のサイズは大きくなってしまいます。出力フォーマットが必要ない場合などは、cputsなどを使用するとサイズは小さくなります。ただし、一度使えばあとは同じですから、あまり神経質になることもないでしょう。これらの似たような関数は種類も多く、使い分け方が最初はよくわからずに戸惑うかもしれませんが、慣れてくるとありがたみがわかってくることでしょう。

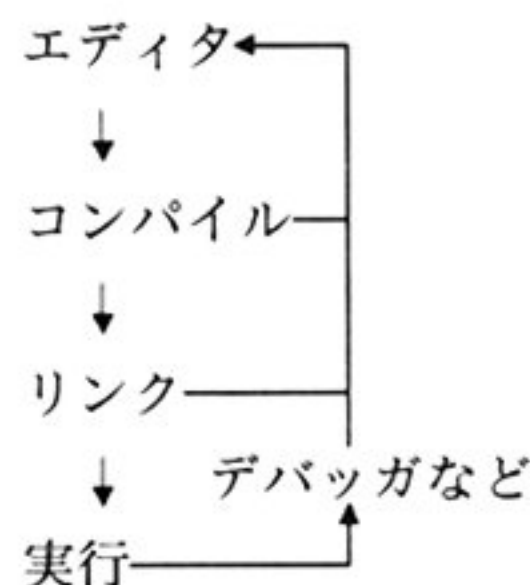
先ほどの文字列の例でも明らかなように、BASICの命令は非常に高機能です。しかしながら、高機能を満足するために言語自体に相当な無理があります。その無理が、実行速度や実行形式のサイズに端的に現われているわけです。もしもあなたが、よりコンピュータを身近に感じたいならば、C言語が絶対にお勧めです。そのかわり、MS-DOSやマシンに関する知識が必要となります。MicrosoftC付属のライブラリだけで、すべてのことは実現できないのですから。

1.3.2 統合環境

最近では、Turbo C、QuickCと統合環境型のCコンパイラが脚光を浴びています。

1. 2. 1でもお話したとおり、コンパイラを使う環境は大変に複雑なものです。今までBASICなどのインタープリタ言語を使用してきた人がC言語に踏み切ることができないのは、コンパイラであるということが大きなウェイトを占めているのではないでしょうか。

今までのMicrosoftCの開発環境といえば、MS-DOS上でエディタ、コンパイラ、リンカ、デバッガ、その他のツールを駆使してのものでした。これらのツールを次々と立ち上げるだけでも、フロッピーベースではとても時間がかかってやっていられないという状況でした。ハードディスクやRAMディスクが一般化した今では、起動自体には問題無いかもしれませんが、これらのツールを使いこなすためにはそれなりに熟練が必要とされてきました。

```
エディタ←──┐
  ↓        │
コンパイル──┤
  ↓        │
リンク─────┤
  ↓   デバッガなど
実行───────┘
```

統合環境とは、これらのMS-DOS上での開発環境をひとつにまとめようというものです。つまり、BASICなどの環境と同じように、ソースを書いたらすぐにコンパイルし、エラーがあればその行へカーソルが飛び修正が可能であり、コンパイルが完了したら、その場で実行も可能というものです。それに加えてQuickCでは、デバッガ機能も一部装備してあり、実行中に変数の内容やアドレスなどを確認できる上、一時停止や再開も可能となっています。

QuickCコンパイラ自体のできは大変良いのですが、以下のようなの問題点もあります。

1. ミディアムモデルのみのサポートである。

なぜミディアムモデルだけなのか不明ですが、実際にはたいていのものはスモールモデルで十分であり、ミディアムモデルでは不必要に大きなサイズになってしまいます。

2. 一部の関数しか使うことができない。

標準で用意されている関数が限定されているため、他からもって来たソースがコンパイルできません。これをコンパイルする方法はあるのですが、手間がかかる上に速度も遅くなってしまいます。

3. エディタをカスタマイズできない。

Turbo Cではカスタマイズできたエディタですが、QuickCでは不可能です。したがって他のエディタに慣れた人にとっては使いにくいものとなっています。

4. メモリを大量に消費する。

環境版の泣きどころがここです。高速化のためにオンメモリでコンパイルするため、メモリを大量に消費し、日本語入力プロセッサなどを使用しているとコンパイルができないことがあります。当然ながら、大きなソースはコンパイルできません。また、子プロセスでの空きも少ないため、各

種のツールを使用できないこともあります。

5. アセンブラ出力ができない。

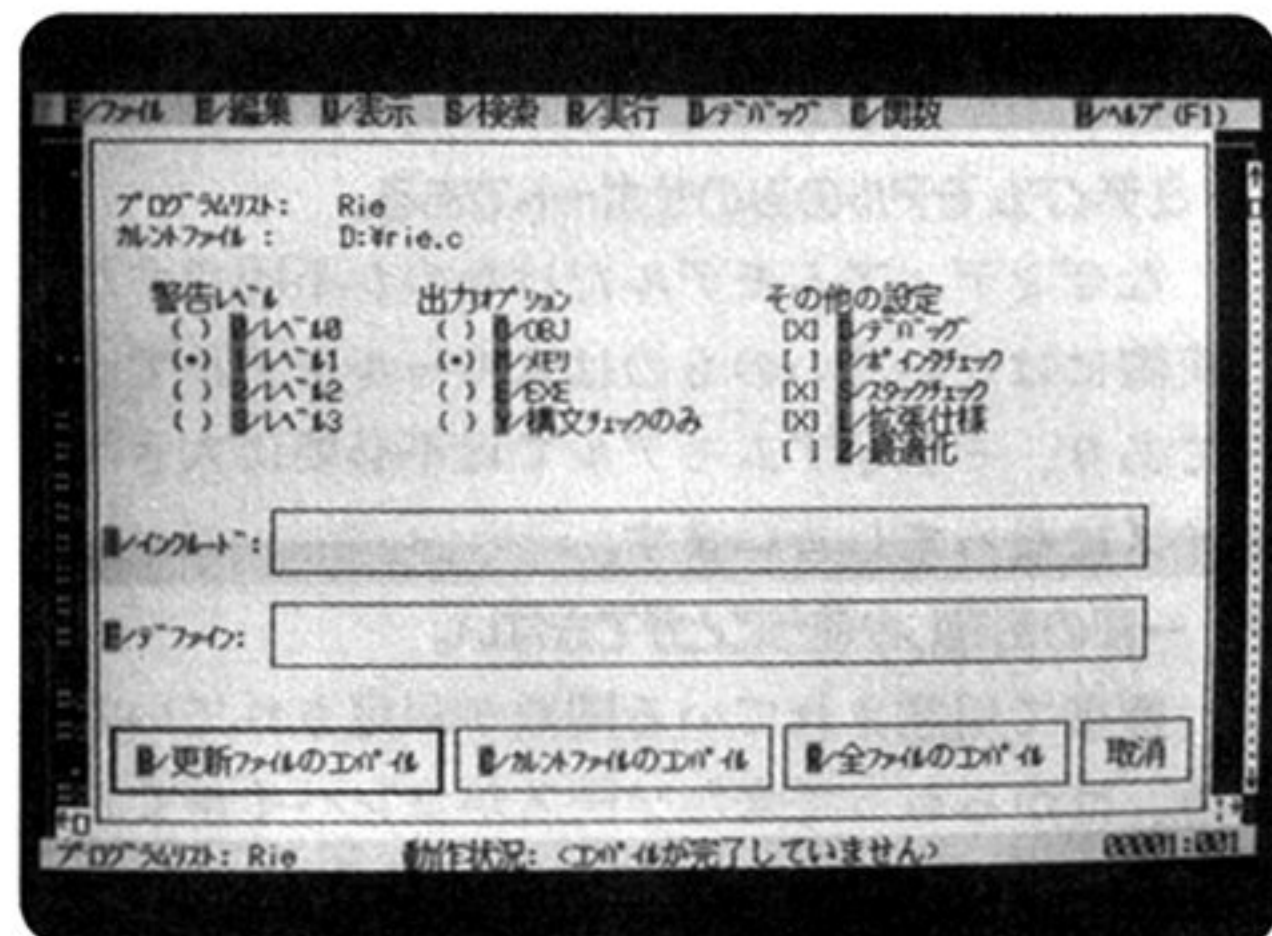
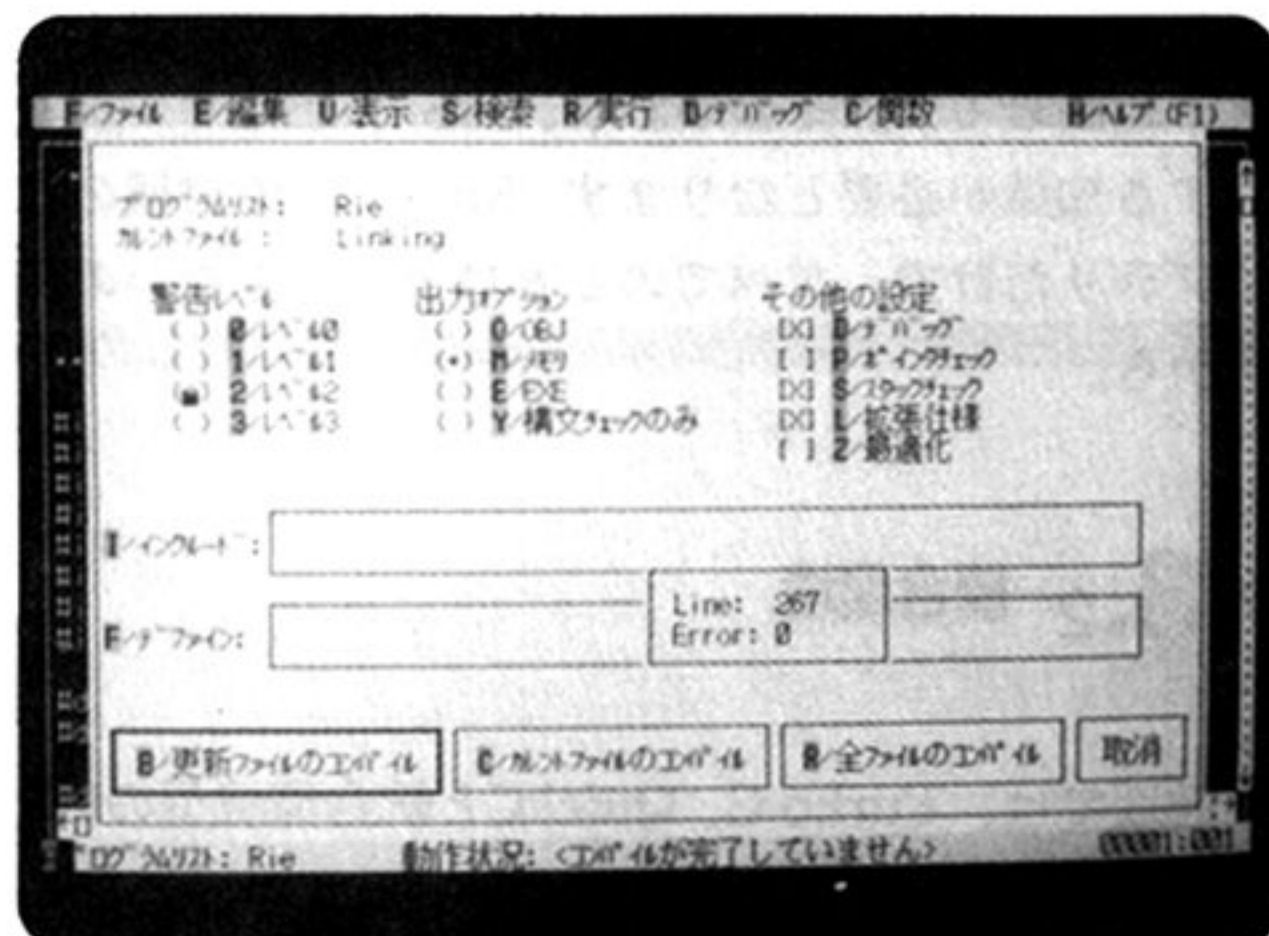
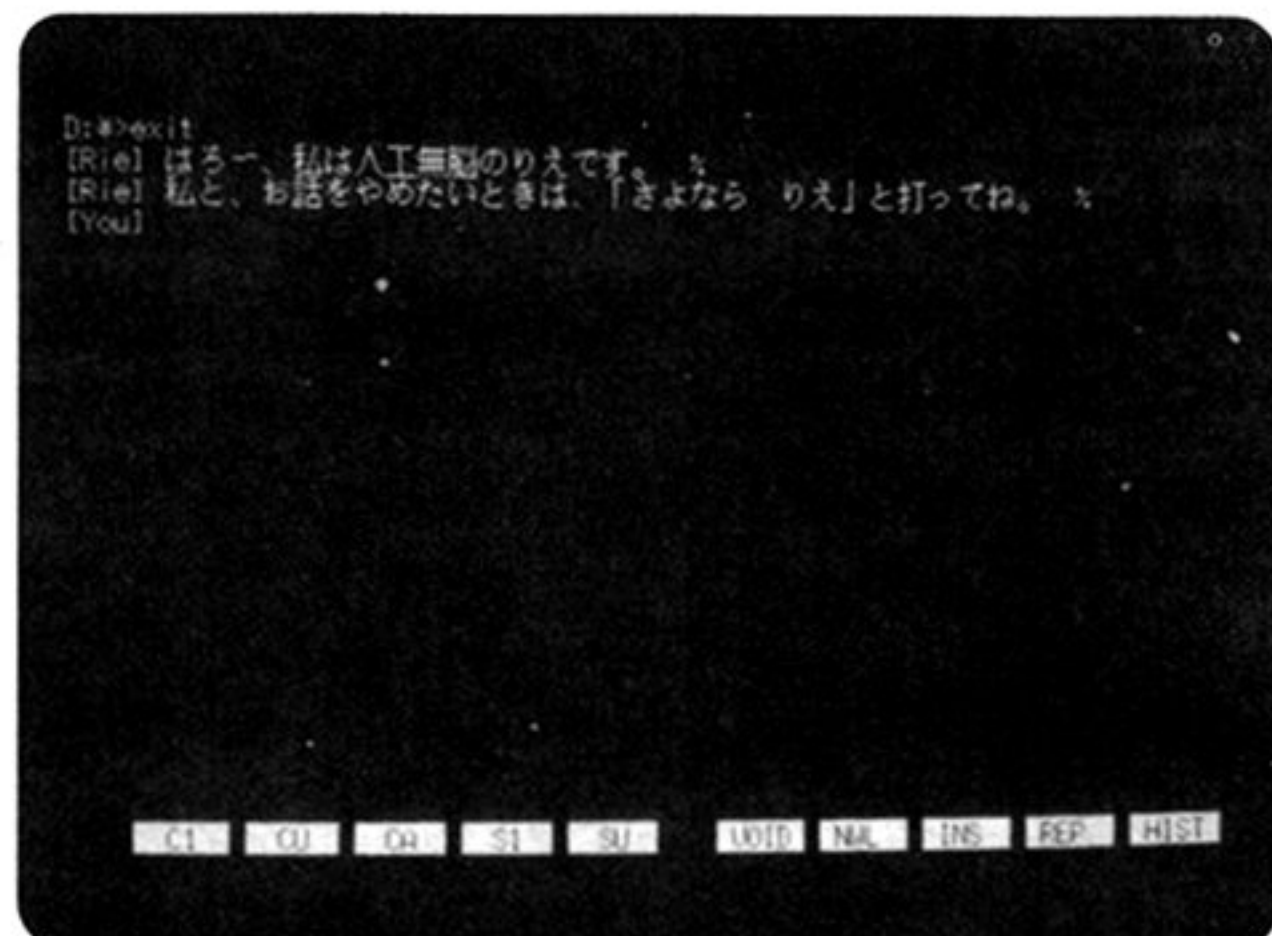
デバッガを内蔵といっても、CPUのレジスタなどの内容を表示しないため、細かいデバッグはできません。コマンドライン版のQCL.EXEでも、アセンブラ出力ができないので、どのようなコードが作成されているのかが、すぐにはわかりません。

結局のところ、QuickCはBASICなどからC言語に移行しようという人をターゲットにしているようです。そのため、ある程度までのレベルまでは取っつきやすく、C言語を収得できるかもしれませんが、この環境に慣れてしまうと、かえってその先に進むのは難しく感じることでしょう。

QuickCもTurbo Cも、環境版以外にもコマンドライン版のコンパイラが付属しています。Turbo Cのものは本格的なものですが、QuickCの場合にはMicrosoftCという親分がいるせいか、サブセット的な色合いが強くなっています。

Turbo　Cでもバージョン2.0からは環境版にもデバッグ機能を持たせていますが、メモリを馬鹿喰いする環境版の特徴はMS-DOSを使う以上はどうにもなりません。

どうしてもC言語が取っ付きにくいものに感じられるならば、環境版の使用もよいかもしれません。しかし、ある程度まで使い込んでくるとどうしても限界がありますから、少々取っ付きにくくても、最初からオーソドックスなMicrosoftCの環境に慣れることをお勧めします。

Quick Cのエディタ画面

同、コンパイル画面

同、リンク画面

同、実行画面

同、デバッグ画面

1.3.3 MicrosoftC vs Turbo C

わが国でも、Turbo C発売当初からMicrosoftCとTurbo Cの対決が話題になりました。Turbo Cは歴史の浅いコンパイラにしては、バグも少なく高速で良質のコードを生成するので、たちまちベストセラーになりました。Turbo Cの一番の特徴は、コンパイル時間が短く、コードも小さいということです。さらに統合開発環境ということを謳い文句に、C言語をより身近なものとしました。機能だけではありません。Turbo Cバージョン1.5の日本での値段は2万円を切るというショッキングなものでした。

MicrosoftCはバージョン4.0からCodeViewというデバッガを標準装備し、LatticeCなどのCコンパイラに打ち勝って(?)、MS-DOSの標準コンパイラになりつつあったのですが、ここでまたしても強敵に出逢ったわけです。

Turbo Cに対抗するため、Microsoftでは環境版を含んだ廉価版としてQuickCを発売しました。QuickCにはTurbo Cの環境版の機能に加え、CodeViewの一部の機能をデバッグ機能として内蔵しました。オンメモリでのコンパイルの速度は、Turbo Cより高速で、デバッグ機能もなかなか有効なものです。

しかし、コマンドライン版のQCL.EXEはコードの質、速度ともにTurbo Cにはかなわないようです。なによりも、アセンブラ出力ができないなどの制限がQuickCをMicrosoftCのサブセットと言わしめています。

MicrosoftCも、バージョン5.0からはライブラリを各モデルにつきひとつとしたおかげで、リンクの速度がアップしました。コンパイル速度もバージョン4.0に比べればアップしているのですが、Turbo Cにはかなわないようです。

実際に、コンパイル速度、実行速度、実行コードのサイズについて、ベンチマークテストをしてみました。(表1.2およびプログラムリスト参照)。

テストプログラムはtest1.cからtest7.cまであり、それぞれ単純ループ、関数呼び出し、メモリアロケーション、ストリング操作、ファイルI/O、ストリームI/O、算術演算です。

実行時間を見てみると、ループについては空ループだったため、CL.EXEは完璧にオプティマイズしています。メモリアロケーションでは、QCL.EXE/CL.EXEがTCC.EXEを大きくリードしていますが、算術計算ではTCC.EXEが抜群に速い結果を出しています。

コンパイル速度を見ると、オプティマイズをしなければ、QCL.EXEはCL.EXEよりも結構速いようですが、TCC.EXEにはかなわないようです。TCC.EXEはオプティマイズのスイッチにほとんど影響されていないように見受けられます。

サイズは、ファイルI/Oと算術計算以外はTCC.EXEの方が概して小さいコードであることがわかります。

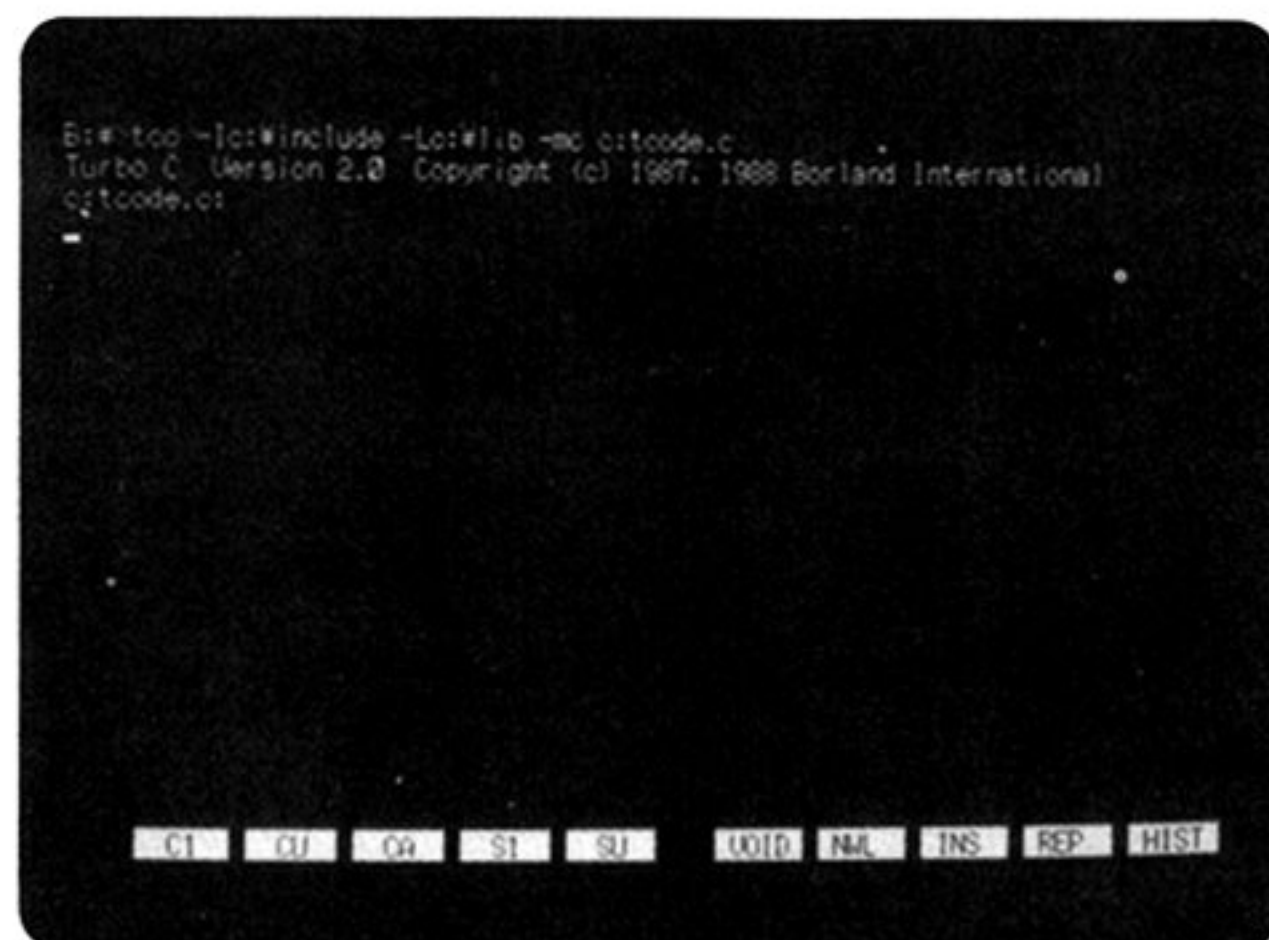
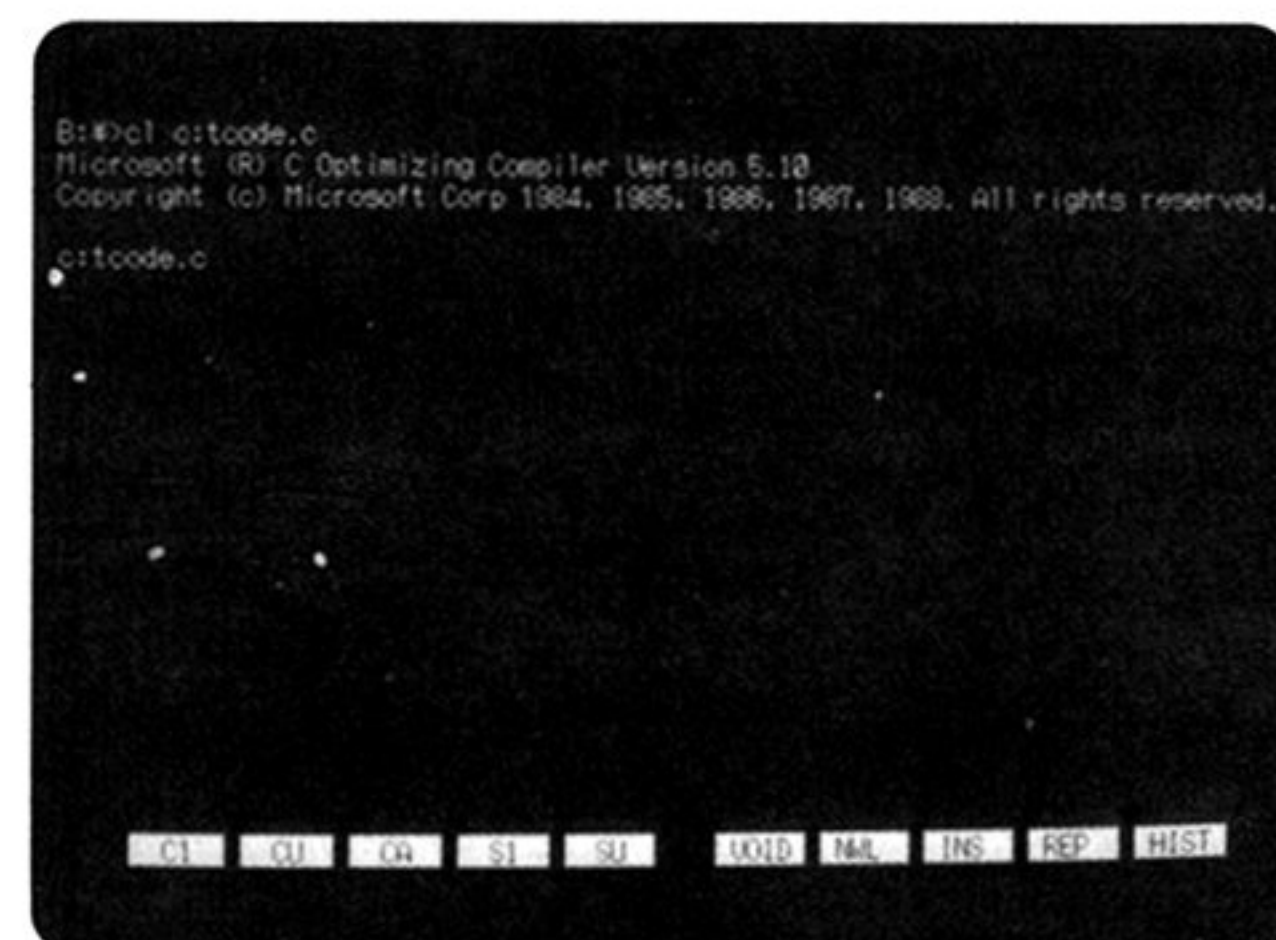
このように、MicrosoftCとTurbo Cを比較する限り、コンパイル速度に関してはTurbo Cの方が速いという結果となりました。しかし、コードの差ではそれぞれに得意分野があるようで、甲乙付け難いものがあります。

Turbo Cの起動画面

Microsoft Cの起動画面

表1.2 【ベンチマークテスト】

コンパイル時間　オプティマイズなし
コンパイル時間　オプティマイズあり
サイズ
実行時間

時間はCRTVの割り込み回数
オプティマイズは、CL,QCLの場合は-Ox、TCCでは-G -O -Z
ミディアム・モデルでコンパイル

	QCL	CL	TCC
sample 1			
compile time	2.25	2.80	1.79
compile time	3.12	2.84	1.79
size	2345	2329	1852
execute time	1.89	0.01	1.88
sample 2			
compile time	2.27	2.87	1.81
compile time	3.13	3.07	1.81
size	2361	2361	1852
execute time	5.55	6.59	4.93
sample 3			
compile time	3.12	3.81	2.46
compile time	4.64	4.08	2.46
size	3819	3819	2060
execute time	3.88	3.76	9.96
sample 4			
compile time	3.17	3.81	2.91
compile time	4.77	3.93	2.91
size	3319	3223	2158
execute time	43.42	43.31	43.75
sample 5			
compile time	3.56	4.87	2.94
compile time	5.33	5.07	2.94
size	4995	4931	7428
execute time	0.76	0.73	0.83
sample 6			
compile time	3.03	3.62	2.41
compile time	4.34	3.70	2.37
size	4585	4553	4070
execute time	6.87	6.86	6.40
sample 7			
compile time	3.81	4.62	2.34
compile time	5.03	4.68	2.34
size	19038	18966	20118
execute time	11.98	11.25	6.84

ベンチマーク・サンプル1～7

```
/*
 * sample 1
 *
 * loop
 *
 */
main()
{
        int i, j;

        for (i = 0; i < 10000; i++)
                for (j = 0; j < 100; j++);
}

/*
 * sample 2
 *
 * function and stack checking.
 *
 */

dummy()
{
        int dummyarry[1];
}


main()
{
        int i, j;

        for (i = 0; i < 10000; i++)
                for (j = 0; j < 100; j++)
                        dummy();
}



/*
 * sample 3
 *
 * memory allocate
 *
 */

#include <stdio.h>
#if defined(__TURBOC__)
#include <alloc.h>
#else
#include <malloc.h>
#endif

main()
{
        int i, j;
        char *p;

        for (i = 0; i < 10000; i++) {
                for (j = 0; j < 10; j++) {
                        if ((p = malloc(10)) == NULL)
                                abort();
                        free(p);
                }
        }
}
```

```
/*
 * sample 4
 *
 * string functions
 */
#include <stdio.h>
#include <string.h>

main()
{
        int i, j;
        static char buf1[10002];
        static char buf2[10002];

        *buf1 = '¥0';
        for (j = 0; j < 10000; j++)
                strcat(buf1, " ");
        strcpy(buf2, buf1);
        if (strcmp(buf1, buf2))
                abort();
}


/*
 * sample 5
 *
 * file I/O
 */

#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#if !defined(__TURBOC__)
#include <sys/types.h>
#endif
#include <sys/stat.h>

main()
{
        int fh;
        int i, j;
        static char buf1[256];
        static char buf2[256];
        char *p;
        char *filename = "testfile.$$$";

        /* set data */
        p = buf1;
        for (i = 0; i < 256; i++)
                *p++ = i;

        /* open file and write */
        if ((fh = open(filename, O_WRONLY | O_BINARY | O_CREAT | O_TRUNC,
                S_IREAD | S_IWRITE)) == -1) {
                perror(filename);
                abort();
        }
        for (i = 0; i < 256; i++)
                write(fh, buf1, sizeof(buf1));
        close(fh);

        /* open file, read and compair */
        if ((fh = open(filename, O_RDONLY | O_BINARY)) == -1) {
                perror(filename);
                abort();
        }
```

```
	for (i = 0; i < 256; i++) {
		read(fh, buf2, sizeof(buf2));
		if (memcmp(buf1, buf2, sizeof(buf2)))
			abort();
	}
	close(fh);
}



/*
 * sample 6
 *
 * stream I/O
 */

#include <stdio.h>

main()
{
	int c;

	for (c = 0; c < 10000; c++)
		putchar('a');
}



/*
 * sample 7
 *
 * math
 *
 */
#include <math.h>

main()
{
	int i;
	double r, d, t;

	r = 0.0;
	for (i = 0; i < 1000; i++) {
		d += sin(r);
		d -= cos(r);
		t = tan(r);
		r += 0.0003;
		t *= d;
		t /= d;
	}
}
```

☆　☆　☆

MicrosoftCでは、cprintfなどのコンソール入出力関数は、マシンに依存しないルーチンを使っていますからサイズも小さく済みますが、Turbo Cではマシン依存型の関数ですから、サイズが大きくなる上に、他のマシンでは動作しないなどということが起きます。いままでのC言語が移植性を重視したものであったのに対して、これは問題ありと言わざるを得ません。

また、MicrosoftCがUNIXのCを意識した構造となっているのに対して、Turbo Cにはかなり独自の部分があります。実際例として、インクルードファイルの名称が大幅に違っています。また、言語仕様もかなり違ってきており、UNIXで開発されたプログラムの移植は、Turbo Cでは容易にはできないでしょう。

これまでの資源を利用するという意味でも、MicrosoftCのコンパチビリティは重要な特徴としてあげられるでしょう。

しかし、一方でマシンを選ぶ設計となってきているのは、MicrosoftCでもTurbo Cでも同じことです。より快適なデバッグ環境を求めると、仕方のないことかもしれませんが、異常に大きなグラフィックライブラリを搭載することは、今まで独自のグラフィックライブラリを作って使用していたプログラマにとっては、無駄なことです。また、参考資料を元に、独力でグラフィックライブラリを作成できないようでは、C言語のプログラマを自認できないでしょう。

このように、MicrosoftCも時代の流れとともに変わりつつあります。しかし、C言語の持つ独特のエレガントさ、宗教にも似た美学は、当分変わることはないでしょう。C言語を使って美しいプログラムを書くことは、単なる自己陶酔ではなく、現代の芸術です。

MicrosoftCで、この最先端の芸術を体験してください。きっとそこに新しい自分を発見できることでしょう。

第2章　MicrosoftCのインストール

インストールとは自分のハードウェア環境に合わせて、購入したソフトウェアが動作するように、自分自身の環境を作成することを言います。MicrosoftCのような言語を快適に使いこなすには、この環境設定はとても重要です。

この章ではインストールの実際と正しく行われたかどうかのチェック、コンパイラの動作、各種オプションの機能、デバッガの使用法などを中心に説明します。

2.1 インストールとは

MicrosoftCは購入したディスクのままでは動作させることはできません。また、何も考えずハードディスクに移したりすると、環境変数が異常に長くなったり、動作はするものの手持ちのハードウェアの性能を充分に引き出せなくなったりもします。

MS-DOSでは、階層ディレクトリをサポートしています。これは、ディスクの容量の増加などにともない、たくさんのファイルがひとつのディスクに入るため、関連のあるファイルをひとつのまとまったグループとして分ける機能です。ファイルのエントリーを見ようとするときに、あまりスクロールさせずに済んだり、ファイル管理の合理化に役に立ちます。

MicrosoftCでは、コンパイルそのものを行う実行ファイルである＊.EXEなどや、Cのプログラム中で、#include <＊.h>として宣言されるインクルードファイル＊.h、コンパイル時に必要になる関数などが入った＊.lib、そして、プログラムのソース＊.cなどは、それぞれ別のディレクトリに入れておくと、開発効率が上がるように設定されています。

そもそも、これらのファイルをすべて合わせると、スモールモデルに必要なファイルだけで2HDのディスク1枚が埋まってしまいます。自分のソースや作ったライブラリ以外は、書き換える必要の無いのが普通ですから、これらのファイルは明確に分け、ごちゃまぜしてにひとつのディレクトリに入れない方が賢明です。

たとえファイルを別のディレクトリに入れておいても、MS-DOSにはサーチパスという機能がありますから、カレントディレクトリに無いファイルでもアクセスすることができます。このサーチパスは、環境変数というMS-DOSの変数に格納されます。同様にして実行ファイルではないライブラリやインクルードファイルのある場所も、環境変数に指定しておけばコンパイラはそのディレクトリを参照します。MicrosoftCは、もともと開発者向けに発売されるものです。かつてのMicrosoftCバージョン4.0以前では、インストールについては「勝手にやってくれ」程度の感覚でパッケージされていました。

しかし、バージョン5.1からは、ライブラリはメモリモデルごとにひとつにまとめられるようになりました。このため、対話型でセットアップできるSETUP.EXEが付属してきます。SETUP.EXEを使用すれば簡単に環境を作ることができます。しかし、実際には自分の環境に合わせて、多少の手直しも必要となります。

Cの全機能を動作させるためには、最低限ハードディスクの空き容量を3Mbytes程度必要とします。しかし自分に必要な機能だけでよいのなら、フロッピーディスク2枚で動作させることもできます。もちろん、RAMディスクやハードディスクを使えば、より効率的に高速実行することができるようになります。ここではMicrosoftCコンパイラのインストール方法について説明しましょう。

2.1.1 必要なもの

Cを動作させるためには、コンパイラやそれが使用するファイル類を適切なディレクトリに格納し、その状況に合わせて環境変数を設定しなければなり

ません。

必要なファイル類としては、コンパイラ・リンカ等の実行ファイル、エラーメッセージファイル、インクルードファイル、ライブラリファイル、デバイスドライバ等が挙げられます。

★実行ファイル

拡張子が .EXE や .COM のもので、MS-DOSのコマンドとして用います。MicrosoftCを使用するためには、CL.EXE, C1.EXE, C2.EXE, C3.EXE, LINK.EXE が必要です。

また、Code View デバッガを使用するためにはCV.EXEが必要となります。

実行ファイルで必須のものは、

cℓ.exe
c1.exe
c2.exe
c3.exe
link.exe
エディタ

などのファイルです。

★インクルードファイル

拡張子が .Hで終わるもので、STDIO.H、STDLIB.Hなど多数あります。これらのファイルはひとつのディレクトリにまとめて格納しておきます。

また、LOCKING.H, STAT.H, TIMEB.H, TYPES.H, UTIME.Hは、その他のインクルードファイルのあるディレクトリに SYS というサブディレクトリを作成し、そこにまとめて格納しておきます。

インクルードファイルは自分で使用するものだけあればよいのですが、実際にはすべてのファイルが必須であると考えておいた方がよいでしょう。

★ライブラリファイル

メモリモデルごとに各ひとつずつのライブラリが必要です。SLIB??.LIB, MLIB??.LIB, CLIB??.LIB, LLIB??.LIBの4種類で、これらはMicrosoftCを購入してきた時点では存在せず、SETUPコマンドによって作成されます。??の部分はSETUP状態により変化します。

ライブラリファイルは、すべて揃えておきたいのですが、ディスク容量の関係で制限のある場合は、SLIBCE.LIBを用意しておけばよいでしょう。

★デバイスドライバ

CodeViewデバッガ(CV)を使用するなら、マウスがあった方が快適です。マウスを使用するにはMOUSE.SYSというデバイスドライバをCONFIG.SYSに登録するか、常駐型マウスドライバMOUSE.COMを使用しなければなりません。

デバイスドライバで特に必須となるものはありません。

2.1.2 フルシステムのインストール

フルシステムをインストールする場合にはMicrosoftCに付属している、SETUP.EXEを使用すると簡単です。このプログラムは対話型になっていて、質問に答えていくことによって、フルシステムを簡単に構築できます。それぞれの質問時には、画面下方に日本語で説明が出ますから、よく読んで質問に答えるようにします。

セットアップ作業に入る前には、SETUP.EXEと同じディスクにあるSETUP.DOCに必ず目を通しておいてください。SETUPを起動すると次のような画面が現われます。

```
--------------------------------------------------------------------------------
Microsoft (R) C Setup Program (C) Copyright Microsoft Corp. 1986, 1988


注意！
セットアップ作業にかかる前に，同じディスクに入っている SETUP.DOC
を必ずお読みください．
SETUP.DOC には，SETUP.EXE の使い方のほかに，メモリモデルや浮動小
数点演算パッケージの組み合せに応じて，それぞれインストールするの
に必要なディスクスペースについても記述されています．

継続するなら C を 終了するなら Q を押して下さい．
--------------------------------------------------------------------------------
```

ここにも書かれているように、SETUP.DOCは必ず読みましょう。読んでいないようでしたら、Qを押してSETUP.DOCを読んでください。

Cを押すと次の画面に進み、各種設定状態について質問されますので、ひとつひとつ答えてください。画面下方には、日本語による解説も表示されますから、よく読んでから、入力しましょう。

```
--------------------------------------------------------------------------------
Microsoft (R) C Setup Program (C) Copyright Microsoft Corp. 1986, 1988

Source of setup files [B:]: C:
Build combined libraries [Y]: Y
Operating Mode: OS/2 Protect Mode [N]: N  OS/2 Real Mode and DOS [Y]: Y
Math Options:  Emulator [Y]: Y  8087 [N]: N  Altmath [N]: N
Memory Models: Small [Y]: Y  Medium [N]: N  Compact [N]: N  Large [N]: N
Use default names: OS/2 Libraries [N]: N  DOS Libraries [Y]: Y
Delete the component libraries when finished [Y]: Y
Include GRAPHICS.LIB in combined libraries [N]: Y
FORTRAN compatability installation files [N]: N
C 4.0 compatable names for SETARGV.OBJ [N]: N
Editor interface: Brief [N]: N  Epsilon [N]: N  QuickC [Y]: Y

Do you want to change any of the above options [Y]: N
--------------------------------------------------------------------------------
```

再び質問されます。ドキュメントファイルや、ブートモジュールのソースファイルをコピーするかどうかの設定です。この設定はMicrosoftCのインストールに直接影響するものではありません。

```
--------------------------------------------------------------------------------
Copy documentation files [Y]: Y
Copy sample C programs [N]: N
Copy C startup sources [N]: N

Do you want to change any of the above options [Y]:
--------------------------------------------------------------------------------
```

インストールをするドライブ・ディレクトリを質問されます。ここではf:を指定しています。ハードディスクのドライブを指定するのが一般的でしょう。ディレクトリ名も好みのものに変更してかまいませんが、このままの設定で行う方がスマートです。

```
--------------------------------------------------------------------------------
Do you want to change any of the above options [Y]:

Directory for Bound executable files [A:¥BINB]: F:¥BINB
Directory for Real Mode (DOS) executable files [F:¥BINR]: F:¥BINR
Directory for Libraries [F:¥LIB]: F:¥LIB
Directory for Include files [F:¥INCLUDE]: F:¥INCLUDE
Directory for Source files [F:¥SOURCE]: F:¥SOURCE

Do you want to change any of the above options [Y]: N

--------------------------------------------------------------------------------
```

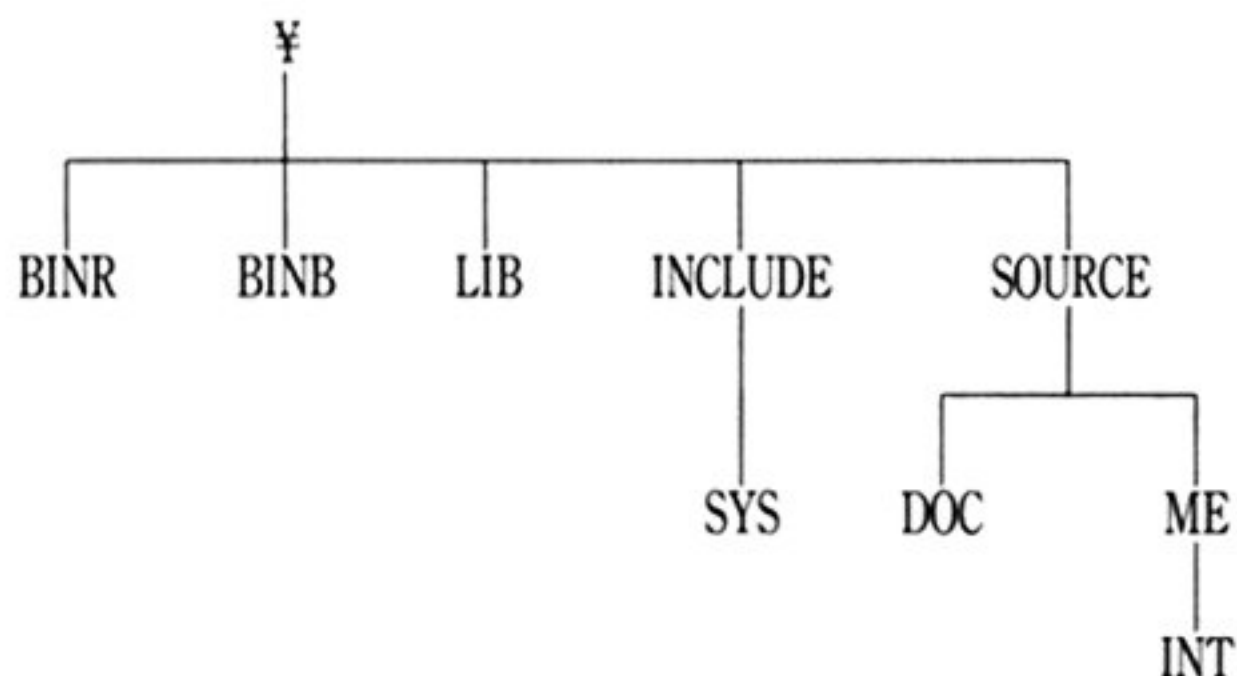
ファイルをすべてコピーし、ライブラリの結合が終了すると、次のように表示されます。何かキーを押すとSETUPの最終画面となります。

```
--------------------------------------------------------------------------------
セットアップの最終作業に入ります.何かキーを押して下さい. . .
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
CONFIG.SYS の例を作成します.
------------------------------------

F:¥RBIN¥NEW-VARS.BAT は,ｺﾝﾊﾟｲﾗが利用する環境変数を設定する AUTOEXEC.BAT の例です
.
AUTOEXEC.BAT に変更するか,これを参考に AUTOEXEC.BAT を書き直して下さい.
MS OS/2 リアルモード と MS-DOS の環境でコンパイラを使用するときにこの
環境の設定が必要です.


F:¥RBIN¥NEW-CONF.SYS は, CONFIG.SYS の例です.コンパイラを使用する時には,これらの
記述が
CONFIG.SYS 中に必要です.これを参考に CONFIG.SYS を修正してください.
MS-DOS の環境で使う時のみ必要です.


Done!
-----


--------------------------------------------------------------------------------
```

以上でSETUPが終了しました。NEW-VARS.BATを実行すると、新しく設定すべき環境変数を設定し、ただちにMicrosoftCが動作する環境になります。

この例では、以下のようなディレクトリが作られます。

```
                 ¥
                 |
 +---------+-----+----+---------+-----------+
 |         |          |         |           |
BINR      BINB       LIB     INCLUDE      SOURCE
                                |           |
                                |       +---+----+
                                |       |        |
                               SYS     DOC       ME
                                                 |
                                                INT
```

以下に、各ディレクトリについて、解説しましょう。

¥BINR
¥BINB

このディレクトリには、＊.EXEのような実行ファイル及びそのエラー内容や、ヘルプ内容の入ったファイルが格納されます。¥BINRにはMicrosoftCのコンパイラ類、¥BINBには付属するユーティリティが入ります。

¥INCLUDE
¥INCLUDE¥SYS

このディレクトリには、Cのプログラム文で最初に #include <stdio.h>などのように宣言される”<” ”>” で囲まれたファイルを格納します。

¥SYSのディレクトリにあるファイルは、XENIXやUNIXのシステムレベルの定数との互換性を持たせたものであり、UNIXに移行時にソースを活かすためのものです。

¥LIB

このディレクトリには、mLIBCE.LIB (mはメモリモデルで、S,M,C,Lがあります)などのようなライブラリを格納します。

¥SOURCE

このディレクトリはワーク用のディレクトリで、自分の作ったソースやライブラリなどを格納します。

では、実際には、各ファイルがどのディレクトリに格納されるかを図2.1のTreeに示します。

```
図2.1              !-- SETUP.EXE
                   !-- CVPACK.EXE
                   !-- EXEHDR.EXE
                   !-- LIB.EXE
                   !-- MAKE.EXE
                   !-- ILINK.EXE
        !-- BINB --!-- CALLTREE.EXE
        !          !-- ECH.EXE
        !          !-- EXP.EXE
        !          !-- MEGREP.EXE
        !          !-- RM.EXE
        !          !-- UNDEL.EXE
        !
        !          !-- LINK.EXE
        !          !-- MOUSE.COM
        !          !-- CL.EXE
        !          !-- CL.HLP
        !          !-- CL.ERR
        !          !-- C1.EXE
        !          !-- C2.EXE
        !          !-- C23.ERR
!-- . --!          !-- C1.ERR
        !          !-- C1L.EXE
        !-- BINR --!-- C3.EXE
        !          !-- CV.EXE
        !          !-- CV.HLP
        !          !-- ERROUT.EXE
        !          !-- EXEPACK.EXE
        !          !-- EXEMOD.EXE
        !          !-- EXEC.EXE
        !          !-- M.EXE
        !          !-- NEW-VARS.BAT
        !          !-- NEW-CONF.SYS
        !
        !         !-- SVARSTCK.OBJ
        !         !-- BINMODE.OBJ
        !-- LIB --!-- SETARGV.OBJ
        !         !-- SLIBCE.LIB
        !
        !                        !-- LOCKING.H
        !                        !-- STAT.H
        !             !-- SYS --!-- TIMEB.H
        !             !          !-- TYPES.H
        !             !          !-- UTIME.H
        !             !-- ASSERT.H
        !             !-- CONIO.H
        !             !-- CTYPE.H
        !             !-- JCTYPE.H
        !             !-- DIRECT.H
        !             !-- DOS.H
        !             !-- ERRNO.H
        !             !-- FCNTL.H
        !             !-- FLOAT.H
        !             !-- IO.H
        !             !-- LIMITS.H
        !-- INCLUDE --!-- MALLOC.H
        !             !-- MATH.H
        !             !-- MEMORY.H
        !             !-- PROCESS.H
        !             !-- SEARCH.H
        !             !-- SETJMP.H
        !             !-- SHARE.H
        !             !-- SIGNAL.H
        !             !-- STDARG.H
        !             !-- STDDEF.H
        !             !-- STDIO.H
        !             !-- STDLIB.H
        !             !-- STRING.H
        !             !-- JSTRING.H
        !             !-- TIME.H
        !             !-- VARARGS.H
        !
        !                       !-- README.DOC
        !            !-- DOC --!-- SETUP.DOC
        !            !
        !-- SOURCE --!                   !-- BRIEF.INI
                     !-- ME --!-- INI --!-- EPSILON.INI
                                         !-- TOOLS.INI
```

ファイルの設定は以上で済みましたが、MicrosoftCは各種の環境変数を設定しないと動作しません。MicrosoftCで使える環境変数には以下のようなものがあります。

★サーチ順指定

PATH
LIB
INCLUDE
TMP

★オプション指定

CL
LINK

これらの環境変数のうちでも、サーチパスはMS-DOSのコマンドラインから"set"を用いて、

```
set lib = f:¥lib
```

のように指定します。また、TMP以外は

```
set lib = f:¥lib;¥mylib
```

などのように";"を用いて、探してくる順を複数指定することもできます。

オプション指定は、コンパイラや、リンカのファイル名やオプションを指定できるもので、

```
set cl = demo.c -c
```

などのように、やはり"set"を用いて指定します。

以下に、それぞれについての詳細を述べます。

2.1.2.1 PATH

PATHは、指定した実行ファイルがカレントディレクトリに無い時に、コマンドインタープリタ(COMMAND.COM)が、どのディレクトリを探すかを決める環境変数であり、MicrosoftC以外でも使われるものです。

現在のカレントディレクトリが、f:¥sourceである時に、

```
set path = a:¥;f:¥bin;a:¥tools;a:¥editor
```

と、指定したとすれば、

```
f:¥source
    ↓
a:¥
    ↓
f:¥bin
    ↓
a:¥tools
    ↓
a:¥editor
```

の順にサーチして行きます。

2.1.2.2 LIB

リンカ起動時に、関数などの入ったライブラリの入ったディレクトリをサーチする順を指定する環境変数です。

指定方法などはPATHと同様です。

2.1.2.3 INCLUDE

コンパイル時に、ヘッダファイルの入ったディレクトリをサーチする順を指定する環境変数です。

指定方法はPATHと同様ですが、¥SYSを指定する必要はありません。その代わり、プログラム側で、

```
#include <sys¥stat.h>
```

のように書かなければなりません。

この時のサーチする順は、

```
b:¥msc¥include
    ↓
b:¥msc¥source
```

となります。

なお、

```
#include "myprog.h"
```

としておけば、カレントディレクトリだけを探しに行きます。

2.1.2.4 TMP

コンパイラが作る作業用のファイルのディレクトリを指定するものです。

このディレクトリは、通常ソースファイルの2倍の容量を必要とします。また、このファイルへのアクセスは頻繁になりますから、RAM-diskがあれば、そのディレクトリを指定するのがよいでしょう。

この環境変数は、他のサーチ順指定と違って、ひとつのディレクトリしか指定できません。

指定方法は

```
set tmp=d:¥
```

などのようにします。

この環境変数が指定されていないとき、コンパイラはカレントディレクトリを作業用ファイルを作るディレクトリとします。

2.1.2.5 CL

環境変数CLは、CL.EXEを起動した時のオプションと同じ内容を設定する時に使います。

サーチ順指定の環境変数と違い、必ずしも設定しなくても構いません。

MS-DOSのコマンドラインから、

```
cl test.c /Am /c
```

と入力したい時に、あらかじめ、

```
set cl = test.c /Am /c
```

のように環境変数にセットしておけば、その後はコマンドラインから単に、

```
cl
```

と入力するだけで同じ結果が得られます。

ここで、注意しなければならないのは、環境変数CLを指定した後で、

```
cl test.c /c
```

などと入力すると、CL.EXEは、

```
cl test.c /Am /c test.c /c
```

と解釈するため、2番めのtest.cをtest.objとしてLINK.EXEに渡し、リードライトモードでオープンされ、test.cを壊してしまいます。

ですから、この環境変数の使用はできれば控えた方が賢明でしょう。

2.1.2.6 LINK

環境変数LINKは、LINK.EXEを起動した時のオプションと同じ内容を設定する時に使います。サーチ順指定の環境変数とは違い、CLと同様で必ずしも設定しなくてもかまいません。

この環境変数についてもCLと同様な注意が必要です。

その他に、CONFIG.SYSの内容についても注意が必要です。

MicrosoftCでは、ファイルを20以上使うことがあるため、CONFIG.SYSの中で、

```
files = 25
```

また、バッファの容量も最低10以上はとった方が円滑なコンパイルができますから、

```
buffers = 10
```

などのように、ファイルおよびバッファを指定します。

環境変数エリアを拡張したい時には、

```
shell=a:¥command.com a:¥ /p /E:20
```

などのように、E:の後にサイズを指定します。何も指定しない時のサイズは10です。

なお、この指定はMS-DOS Ver.3.1の場合です。Ver.3.3では指定法が異なります。

2.1.3 フロッピーディスク2枚で動作させるインストール

2. 1. 1で述べたように、MicrosoftCがとりあえず動作するために必要なファイルのみを集めれば、フロッピーディスク2枚に収めることもできます。これはあくまでも「とりあえず」的な方法ですので、ラップトップコンピュータで使用する場合に限ったものと考えてください。ここで述べる設定は決して快適な環境とは言えません。MicrosoftCを本格的に使うには、やはりハードディスクが必要でしょう。

フロッピーディスク2枚で動作させると必要最低限のファイルしか置けません。前述の「必須」とされているファイルのみを使用することになります。

そのようなインストール例を以下に示します。

```
(フロッピーAに入れるファイル)
A:¥COMMAND.COM
A:¥CONFIG.SYS
A:¥AUTOEXEC.BAT

A:¥BIN¥C1.ERR
A:¥BIN¥C1.EXE
A:¥BIN¥C2.EXE
A:¥BIN¥C23.ERR
A:¥BIN¥C3.EXE
A:¥BIN¥CL.ERR
A:¥BIN¥CL.EXE
A:¥BIN¥CL.HLP
A:¥BIN¥LINK.EXE

A:¥INCLUDE¥*.H
A:¥INCLUDE¥SYS¥*.H

A:¥LIB¥SLIBCE.LIB

(フロッピーBに入れるファイル)
B:¥EDITOR¥MIFES.EXE
B:¥EDITOR¥MIFES.HLP
B:¥SRC¥
```

2.1.4 インストールのチェック

正しくインストールされたかチェックするため、簡単なプログラムを書いてコンパイルさせてみましょう。2. 2にあるtest.cをエディタなどで入力してください。入力後、cl test.cを実行します。正常にコンパイル・リンクされたならば、ほぼインストールは成功していると思っていいでしょう。

clコマンドを実行した際に、「コマンドまたはファイル名が違います」とのエラーが発生した場合、実行ファイルが正しくコピーされていないか、環境変数PATHの設定が間違っています。c1.exe,c2.exe等の実行ファイルが格納されているディレクトリを確認してください。正しくcl.exeがコピーされているでしょうか。問題が無ければ、PATHコマンドを実行してコマンドサーチパスを表示させてみてください。実行ファイルが格納されているディレクトリがPATHに正しく追加されているでしょうか。このエラーが発生するのは、ほとんどこの2点の間違いと思われます。

似たような間違いによって発生するエラーがあります。

```
Command line warning D4005 : could not execute 'c1.exe';

Command line warning D4005 : could not execute 'c2.exe';

Command line warning D4005 : could not execute 'c3.exe';
```

の3種類です。これはc1.exeやc2.exe、c3.exeが実行ファイルのディレクトリに正しく格納されていないことを示唆しています。実行ファイルのディレクトリを確認してください。

```
test.c(2) : fatal error C1015: cannot open include file 'stdio.h'
```

といったエラーが発生する場合は、インクルードファイルが正しくコピーされていないか、環境変数INCLUDEの設定の誤りでしょう。インクルードフ

ァイルのディレクトリを確認し、SETコマンドで環境変数INCLUDEがそのディレクトリを指しているか確認してください。

LINK : warning L4051 : SLIBCE.LIB : cannot find library

　これも同様に、ライブラリファイルの設定が正しく行われていないために発生します。ライブラリファイルのディレクトリを確認し、環境変数LIBがそのディレクトリを指しているかを確認してください。

Command line fatal error D1001 : could not execute 'A:¥BIN¥c1.exe'

　このエラーはメモリが不足している場合に発生します。MicrosoftCを実行するためにはRAMが640KB必要です。640KB実装しているのにかかわらずこのエラーが発生する場合は、SWITCHコマンドでメモリ空間が640KBに設定されているかどうか確かめてください。メモリ空間が640KBに設定されていてもこのエラーが発生するならば、デバイスドライバや常駐プログラム等によってフリーメモリ空間が少なくなっているものと思われます。不必要なデバイスドライバや常駐プログラムを外してシステムをリセットし、再試行してください。

fatal error C1041 :
fatal error C1041 : cannot open compiler intermediate file - no more files

　これらのエラーは、CONFIG.SYSに設定されているFILES=の数値が小さすぎることを示します。FILES=25程度に設定しておくことをお勧めします。

2.2 コンパイラの動作

Cコンパイラはで書かれたプログラムを機械語に変換するものです。画面に文字列を表示するだけのプログラムtest.cを作成したとします。

```
/**********test.c***********/
#include <stdio.h>

void main(void)
 {
    printf("test.c¥n");
}
/*************************/
```

このプログラムをコンパイルすると、

```
A>cl test.c
Microsoft (R) C Optimizing Compiler
Version 5.10
Copyright (c) Microsoft Corp 1984, 1985, 1986,
1987, 1988. All rights reserved.
test.c

Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.
All rights reserved.

Object Modules [.OBJ] : TEST.OBJ
Run File [TEST.EXE] : TEST.EXE /NOI
List File [NUL.MAP] : NUL
Libraries [.LIB] :
```

と表示され、コンパイルが終了します。コンパイラによって実行ファイルtest.exeが作られました。以後は test [return] と入力すれば画面にtest.cという表示を行います。

このようにtest.cという人間にわかりやすいC言語のソースを機械が実行可能な形式であるtest.exeにまで変換する一連の作業が、clコマンドによって実行されます。

以前流行し、どのコンピュータにも付属していたBASICなどという言語では、プログラムを入力し終った時点でRUNと入力すれば実行される形式をとっていました。C言語ではコンパイルしてから実行可能なファイルが作られ、それを実行するという形式です。このような言語をコンパイラ言語と呼び、BASICのようにすぐに実行できる言語をインタープリタ言語と呼びます。インタープリタ言語では、すぐ実行できるという利点はありますが、実行速度はコンパイラの方がかなり上回ります(10倍以上でしょう)。

C言語で書かれたプログラムは即実行できるものではないと述べました。コンパイルという作業を経なければならないからです。コンパイルにはそれなりの時間がかかります。C言語で書かれたソースプログラムが大きくなればなるほど、コンパイルに要する時間も長くなります。巨大なプログラムのごく一部を書き直したからといって全部をコンパイルし直さねばならないとしたら大変時間がかかってしまいます。

リンカ(LINK)はこのような問題を解決するツールです。複数のソースプログラムで作られたプログラムをひとつにまとめて実行形式のファイルにします。たとえば、

```
/**********test2.c**********/

void main(void)
 {
        hello("MANAMI");
        hello("REIKO");
}
/*************************/
```

というソースファイルtest2.cと

```
/**********test3.c**********/
#include <stdio.h>
void hello(char *name)
 {
        printf("Hello > %s¥n", name);
}
/*************************/
```

というソースファイルtest3.cがあるとします。これらのファイルをコンパイルして、実行形式ファイルを作成します。

```
A>cl test2.c test3.c
Microsoft (R) C Optimizing Compiler
Version 5.10
Copyright (c) Microsoft Corp 1984, 1985, 1986,
1987, 1988. All rights reserved.
test2.c
test3.c

Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.
All rights reserved.

Object Modules [.OBJ] : TEST2.OBJ +
Object Modules [.OBJ] : TEST3.OBJ
Run File [TEST2.EXE] : TEST2.EXE /NOI
List File [NUL.MAP] : NUL
Libraries [.LIB] :

A>test2
Hello > MANAMI
Hello > REIKO
```

コンパイラが、

```
test2.c
test3.c
```

というメッセージを出力してふたつのソースプログラムがコンパイルされたことがわかります。これで複数のソースプログラムからひとつの実行型ファイル(test2.exe)が作成されました。ここで、test2.cに書かれている

```
hello("MANAMI");
```

を

```
hello("麻奈美");
```

に書き直したとします。このような場合test3.cは変更されていないので再度コンパイルする必要はありません。test2.cのコンパイルのみを行うには次のようにします。

```
A>cl test2.c test
Microsoft (R) C Optimizing Compiler
Version 5.10
Copyright (c) Microsoft Corp 1984, 1985, 1986,
1987, 1988. All rights reserved.
test2.c

Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.
All rights reserved.

Object Modules [.OBJ] : TEST2.OBJ +
Object Modules [.OBJ] : TEST3
Run File [TEST2.EXE] : TEST2.EXE /NOI
List File [NUL.MAP] : NUL
Libraries [.LIB] :
```

コンパイラの出力メッセージには

```
test2.c
```

とのみ表示されました。test3.cはコンパイルされていません。しかしtest2を実行させてみると、

```
A>test2
Hello > 麻奈美
Hello > REIKO
```

のように正常に書き変わっています。このように変更していない部分はコンパイルしないでも済むので、全体が巨大なプログラムであっても一部を変更したときにやらねばならないコンパイル作業は、変更した部分だけということになります。このようなプログラムは、モジュール分割されたプログラムと呼ばれ、Cでの開発の効率を上げる常識的な手法です。

モジュール分割の意義はコンパイル速度のみにあるものではありません。複数の人がプログラムを作る場合などにも、とても有効です。

しかし、最大のメリットはソースプログラムが見やすくなるという点でしょう。数画面もあるファイルから変更したい目的の部分を探すのは大変なことです。プログラムを機能別に細分化しておけば目的箇所は素早く見つかり、プログラムの修正は速くなります。これは極端な例ですが、人によってはひとつのファイルにひとつの関数を書いて、その関数が

一画面に収まるようにするといった分割手法をとっている場合もあります。

では、なぜコンパイルをせずに実行形式のファイルができたのでしょうか。Cコンパイラはソースプログラムtest2.c, test3.cから直接test2.exeを作成したわけではありません。test2.obj, test3.objというオブジェクトファイルを作り、リンカ(LINK)がtest2.obj、test3.obj、ライブラリをひとつにまとめてtest2.exeを作成したのです。

また、同時に注目してほしいのは、test2.cの中でいきなりhello()という関数を使用しています。この関数はtest2.cでは何の関数であるかわかりません。関数リファレンスの章を読んでもhello()については何も述べられていないでしょう。このhello()という関数はtest3.cの中で定義されている外部関数です。このような関数を呼び出せるようにするのもリンカの役割です。

先ほどのtest.cでもリンカは使われていました。printfという部分です。ここではMicrosoftCのライブラリファイルの中にある関数printfをオブジェクトファイルと結合する作業が行われました。

Cコンパイラはprintfが何物かということをまったく理解していません。これは大変重要なことです。PascalやBASICなどの他の言語では、Writeln('string'); だったらば文字出力、INPUT Aだったらば数値入力と決められていました。これは言語の仕様として入出力関数を定義しているためで、コンパイラやインタープリタの動作を不必要に複雑にしていました。Cではコンパイラの言語仕様とライブラリ関数の仕様を明確に区分しており、コンパイラ自体は大変コンパクトに実現されています。可能ならば自分でまったく違う仕様のprintf関数を作成して、そちらを使うことすらできるのです。

2.3 CL、Link、その他

ここでは、MicrosoftCの中核をなすコンパイラ、リンカについて解説します。

2.3.1 CLコマンド

clコマンドはソースファイルのコンパイルを行い、必要に応じてLINKを起動します。

(1) CLの起動方法

CL.EXEは、以下のようなパラメータを付けて起動します。

```
cl [option] file file [file ...]  [/link libfile [libfile ...]]
```

optionはコンパイルオプションで、詳しい内容については後述します。file にはソースファイル名またはオブジェクトファイル名を書きます。ソースファイルに対しては .c の拡張子を必ず付加しなければなりません。拡張子が無いファイルや、拡張子が .c 以外のファイルはオブジェクトファイルであると見なされ、コンパイルをせずにリンクされます。

```
cl test.c test2.obj teset3
```

このように書いた場合、test.c, test3をコンパイルし、test.obj test2.obj test3.objをリンクします。

(2) CLのオプション

オプションがわからなくなってしまった場合のために、/help というオプションが用意されています。このオプションさえ憶えていれば簡単な説明が表示されるので、マニュアルと首っぴきというようなことにはならないでしょう。

最低限cl /helpだけは憶えておきましょう。

なお、スイッチキャラクタとして'/'を用いて説明しておりますが、'-'でも同等の動作をします。たとえば、cl /help は、cl -helpと同義です。 cl /c test.c は、cl -c test.c と同義です。また、オプションは大文字小文字を区別しますので注意が必要です。

【clコマンドオプション一覧】注：★の付いたものがデフォルトです

オプション	内容
/AS	スモールモデル★
/AC	コンパクトモデル
/AM	ミディアムモデル
/AL	ラージモデル
/AH	ヒュージモデル
/O	オプティマイズを行う
/Oa	別名定義を無視する
/Od	オプティマイズを禁止する
/Oi	インライン展開を行う
/Ol	ループオプティマイズを行う
/On	危険なオプティマイズを禁止
/Op	浮動小数点演算結果の整合性の確保
/Or	インラインリターンを禁止
/Os	サイズを縮小を優先
/Ot	実行速度を優先★
/Ox	最大限のオプティマイズを行う
/G0	8086命令を用いる★
/G1	80186命令を用いる
/G2	80286命令を用いる
/Gm	文字列をconstセグメントに置く
/Gc	パスカル呼びだし形式
/Gs	スタックチェックを行わない
/Gt数値	データサイズのしきい値
/Faファイル名	アセンブラリスティングファイル
/Fcファイル名	混合リスティングファイル
/Feファイル名	実行ファイル
/Flファイル名	オブジェクトリスティングファイル
/Fmファイル名	リンクマップファイル
/Foファイル名	オブジェクトファイル
/Fsファイル名	ソースリスティングファイル
/Sl数値	1行の文字数
/Sp数値	1ページの行数
/St文字列	タイトル文字列
/Ss文字列	サブタイトル文字列
/C	コメントを残す
/D名前=文字列	マクロ定義
/E	プリプロセッサ出力
/EP	#lineを出力しないプリプロセッサ出力
/Iディレクトリ	インクルードファイルのサーチパス
/P	ファイルへのプリプロセッサ出力
/U名前	定義済みのマクロを解除する
/u	すべての定義済みのマクロを解除する
/X	環境変数 INCLUDE を無視する
/Za	拡張言語仕様を無効にする
/Zd	行番号情報を付加する(symdeb用)
/Ze	拡張仕様を有効にする★
/Zg	プロトタイプ宣言を出力する
/Zi	シボリックデバッグ情報を出力する(cv用)
/Zl	デフォルトライブラリ情報を出力しない
/Zp数値	構造体メンバの境界アドレス
/Zs	文法チェックのみを行う
/FPa	簡易浮動小数点ライブラリ
/FPc	エミュレータライブラリコール
/FPc87	87ライブラリコール
/FPi	エミュレータのインライン展開★
/FPi87	87コードのインライン展開
/c	コンパイルのみを行う
/H数値	外部定義名の長さ
/J	charを符号なしとする
/Tcファイル名	拡張子にかかわらずコンパイルする
/V文字列	文字列を埋めこむ
/W数値	ワーニングレベル
/F数値	スタックサイズ
/Lc /Lr	リアルモード
/Lp	プロテクトモード
/link	リンカのオプション・ライブラリ

★メモリモデルに関するオプション

メモリモデルにはS、C、M、L、Hの5種類があります。通常のプログラミングではスモールモデルだけで充分ですが大きなプログラムを作成する場合、C、M、L、Hが必要となってきます。

/AS スモールモデルを指定します。

スモールモデルは全メモリモデルのなかで実行速度が一番高速になります。ただし、コードサイズ、データサイズともに64KB以内でなければなりません。普通に使用するぶんにはスモールモデルのみで充分ですが、大きなプログラムを書く場合には他のモデルを使わねばなりません。

他にメモリモデルに関するオプションが何も指定されていない場合、このオプションが指定されたものと見なされます。

/AC コンパクトモデルを指定します。

コンパクトモデルは、大きな配列を用いるようなプログラムに使用します。コンパクトモデルではコードサイズは64KB以内である必要がありますが、データサイズは64KBを越えても構いません。ただし個々の配列が64KBを越えることはできません。そのような場合にはヒュージモデルを使用してください。

/AM ミディアムモデルを指定します。

ミディアムモデルはコード部分が大きく、データ部分は64KB以内に収まるようなプログラムに用います。コード部分には大きさの制限がありません。ただし、個々のソースファイルから作られるコード領域が64KBを越えるものはコンパイルできませんので、このような場合複数のモジュールに分割する必要があります。ひとつの関数が64KBを越える場合は8086系のCPUでは実行不能ですので設計をやり直さなければなりません。

/AL ラージモデルを指定します。

ラージモデルはコード・データ共に大きなプログラムに使用します。コード・データ共に制限はありませんが、/ACで述べたように、個々の配列が64KBを越えることはできません。また、/AMで述べたように、個々のソースファイル内でのコードサイズが64KBを越えることはできません。64KBを越えるような配列を使用しなければならない場合はヒュージモデルを使用してください。64KBを越えるような関数は再設計が必要です。

/AH ヒュージモデルを指定します。

ヒュージモデルは64KB以上の配列を操作するとき等に使用します。コード部分については、ミディアム・ラージと同じ制限を受けます。

★最適化に関するオプション

最適化とはコンパイラが生成するコードをより小さく、より速く実行できるように加工することです。より小さく最適化したコードは少ないメモリで実行できますし、より速く最適化したコードは実行時間が短くなります。速くて小さいコードが一番良いのはもちろんのことですが、使用目的に応じて最適化することになります。

/O 最適化を行います。
/Od 最適化を行いません。

/Oは/Otと同義です。/Odを指定しない場合は/Otが指定されたことになります。/Odは最適化を行わない指定です。アセンブラリストを出力してどのようなコードが出力されているか調べる場合には、/Odを指定すると読みやすくなります。

/Oa 別名定義を無視します。

このオプションは複数のポインタが同一領域を参照している可能性が無いことを前提とします。期待しない結果を出力する可能性があるので、なるべく使用しない方がよいでしょう。別名定義を行っているプログラムの例を{s_alias.c}で示します。このようなプログラムをコンパイルした場合、不都合が生じます。たとえば、以下のような場合です。

```
/*
 *s_alias.c
 *
 */0a  オプションにより不都合が生じるプログラ
       ムの例
 *
 */

#include <stdio.h>

void main(void)
{
        int i;
        int *p, *q;

        p=&i;
        q=&i;
        *p=10;
        *q=20;

        printf("%d %d¥n", *p, *q);
}
```

/Oi 関数のインライン展開を行います。

メモリ操作関数、ビット操作関数、I/O関数、8087でサポートされている数学関数は、関数呼出しではなく直接コード生成した方が効率がよくなります。このオプションを指定することによって{inline.lst}で示す関数がインライン展開されるようになります。この指定は、#pragma intrinsic(func, ..)や#pragma function(func, ...)によってソースコード中に書き込むことができ、コマンドラインの指定より優先します。#pragma intrinsicで列挙された関数はインライン展開されます。#pragma functionで指定された関数はインライン展開されません。実行速度を優先させる場合は、すべての関数をインライン展開しても問題ありません。コードサイズを優先させる場合、メモリやストリング関数ではインライン展開を行うとかなりサイズが大きくなってしまいますから、#pragmaによって個々に指定する方がよいでしょう。

```
#include <stdio.h>
#include <math.h>

#pragma intrinsic(sin, cos)

void main(void)
{
        double r;

        r=sin(0.5)*cos(0.5);
        printf("%f¥n", r);
}
```

/Ol ループの最適化を行います。

ループの最適化を行います。空ループが無視されるようになるので、周辺機器とのタイミング合わせ等の意識的なウェイトが行われなくなる可能性があり、注意が必要です。ループの最適化によってループが消去されてしまう例を以下の{s_loop.c}で示します。

```
void main(void)
{
        int i;
        int j;

        for (i=0; i<32000; i++)
            j=10;
}
```

この例では、ループがまったく無くなり、j = 10；という文のみが残ります。同様の動作をpragmaで局所的に実現することができます。

書式

```
#pragma loop_opt(on)
#pragma loop_opt(off)
```

/Or インラインリターンの禁止

関数中でreturnが行われたときに、通常は、

```
        mov   sp, bp
        pop   bp
        ret
```

のコードが生成されます。このオプションを指定す

ることによって、関数最後へのジャンプ命令が用いられるようになり、コードサイズが小さくなります。ただし実行速度は/Orを指定しない場合の方が速くなります。コードサイズを小さくしたい場合に使用してください。

/Os コードサイズ優先の最適化
/Ot 実行速度優先の最適化

/Osではなるべく小さな実行ファイルを作るように機能します。/Otではなるべく速いプログラムを作ります。一般にサイズの小さいプログラムは速いプログラムと思われがちですが、速度を犠牲にして小さくまとめるといった最適化の方法もあります。スモールモデルに収まらなくなったプログラムも、/Osを使用すればスモールモデルに収まるケースもあります。通常、ミディアムモデルで/Otを使用するよりもスモールモデルで/Osを使用した方が実行速度は速いものとなります。このような面も考慮して使い分けてください。

/Ox 最大限の最適化

/Oa /Oi /Ol /Ot /Gs が同時に指定されたものと見なします。/Gsは後述のスタックチェックの制止です。この指定は予期せぬ動作をもたらすことがありますので、使用には充分注意が必要です。それぞれの項目についての注意を参照してください。

★コード生成に関するオプション

/G0 8086コードを生成します
/G1 80186コードを生成します
/G2 80286コードを生成します

使用するCPUに合わせて指定します。どれも指定されない場合は、/G0が指定されたものと見なされます。80186・80286が搭載されているマシンでは、/G1, /G2を指定すると、より高速に実行することができますが、80186・80286が搭載されていないマシンでの実行が不能になってしまいます。自分でしか使用しないプログラム以外には/G1, /G2を指定しない方がよいでしょう。また、PC-9801Vシリーズに採用されているNEC製のCPU・V30は80186のほとんどのコードを実行することができますから、/G1でコンパイルされたコードも実行可能です。

/Gm 文字列定数を定数領域に置きます

通常、文字列は_DATAセグメントに置かれますが、このオプションを指定することにより、CONSTセグメントに置かれるようになります。

/Gc パスカル形式の関数呼び出し

マイクロソフトパスカルと共通の関数呼び出し形式を使用します。マイクロソフトパスカルで使用する関数をC言語で記述したい場合に有効です。

/Gs スタックチェックの停止

auto変数の領域を確保する際にスタックチェックを行わなくなります。通常スタックチェックは、関数の初めに行われるので、関数を呼び出すごとにスタックチェックが行われることになります。このオプションを指定するとスタックチェックを行わなくなるため、実行速度、コードサイズ共に向上します。しかし、スタックオーバーフローした時点での強制終了が行われなくなりますので、スタック容量の管理はプログラマが責任を持たねばなりません。このオプションは動作が充分に確認された段階で付けることをお勧めします。同様の動作をpragmaで局所的に実現することができます。

書式
```
#pragma check_stack(on)
#pragma check_stack(off)
```

/Gt データサイズのしきい値の設定

FAR領域に配置するデータブロックの容量を指定します。コンパクトモデル・ラージモデルを使用したからといって、すべてのデータがFAR領域に置かれるわけではありません。256バイト未満のデータブロックはNEAR領域(DGROUP)に置かれ、それ以上の配列はFAR領域に置かれます。NEAR領域のデータはFAR領域のデータよりも高速にアクセスできます。コンパクトモデル・ラージモデルを使用しているのにもかかわらず、DGROUPがオーバーフローした場合には/Gtオプションで適正値を設定し

てください。

★出力ファイルに関するオプション

コンパイラは .objのファイルを出力し、リンクして .exe のファイルを生成します。これらのオプションは .obj、.exe のファイル名を変更したり、リスティングファイルの出力を行わせるものです。

/Fa アセンブラリストファイルの出力

アセンブラ出力を行います。/Faのみを指定した場合、ソースの .c ファイルの拡張子が .asm に変わったファイルが出力されます。/Fa [filename] のように、ファイル名を指定した場合には指定されたファイル名で出力を行います。拡張子 .asm は省略可能です。

ここで出力されたファイルはマイクロソフトマクロアセンブラ（MASM）によりアセンブルすることができます。コンパイラが生成するコードを読んだり、出力されたコードをアセンブラレベルで変更する時に使用します。

/Fc 混合リスティングの作成

Cのソースコードと、出力コードが混在した形のリスティングファイルを出力します。/Fcのみを指定した場合、ソースの .cファイルの拡張子が .cod に変わったファイルが出力されます。/Fc [filename] のように、ファイル名を指定した場合には指定されたファイル名で出力を行います。拡張子 .cod は省略可能です。ソースコードと出力されたコードの比較を行いたい場合等に使用します。/Fcオプションで出力されたコードはMASMでアセンブルすることができません。アセンブルを行いたい場合には/Faオプションを使用してください。

/Fe 実行ファイル名の指定

実行ファイルのファイル名を変更します。/Feオプションを指定しなかった場合、ソースの .cファイルの拡張子が .exe に変わったファイルが出力されます。/Fe [filename] のように、ファイル名を指定した場合には指定されたファイル名で出力を行います。拡張子 .exe は省略可能です。

/Fl 出力コードリスティングの作成

出力コードのリスティングファイルを出力します。/Flのみを指定した場合、ソースの .cファイルの拡張子が .cod に変わったファイルが出力されます。/Fl [filename] のように、ファイル名を指定した場合には指定されたファイル名で出力を行います。拡張子 .cod は省略可能です。出力されるリスティングは/Fcの出力からソース情報を除いたものとなります。

/Fm リンクマップファイルの作成

マップファイルを出力するように指定します。/Fmのみを指定した場合、ソースの .cファイルの拡張子が .map に変わったファイルが出力されます。/Fm [filename] のように、ファイル名を指定した場合には指定されたファイル名で出力を行います。拡張子 .mapは省略可能です。このオプションはclコマンドが処理するのではなく、clコマンドがLINKを呼び出すときのオプションとして処理されます。詳しくはLINKの/MAPオプションを参照してください。シンボリックデバッガ(SYMDEB)を用いてデバッグを行う場合、/Zd /Od と共に指定しなければなりません。

/Fo オブジェクトファイル名の指定

オブジェクトファイル名を変更します。何も指定しなかった場合、ソースの .cファイルの拡張子が. obj に変わったファイルがオブジェクトファイルとして出力されます。/Fcでファイル名を指定した場合には、指定されたファイル名で .objファイルの出力を行います。拡張子 .objは省略可能です。カレントディレクトリ以外にオブジェクトファイルを置きたいときなどに有効です。

/Fs ソースリスティングの作成

ソースファイルリスティングを出力します。/Fsのみを指定した場合、ソースの .cファイルの拡張子が .lstに変わったファイルが出力されます。/Fs [filename] のように、ファイル名を指定した場合には指定されたファイル名で出力を行います。拡張子 .lstは省略可能です。

★ソースリスティングオプション

これらのオプションはソースリスティング出力オプション/Fsが指定された場合に有効です。ソースリスティングの横文字数、行数、タイトル行に書き込まれる文字列等を指定します。

/Sl<columns> 1行の文字数の設定

1行の文字数を指定します。79から132までの数値で指定します。範囲外の数を設定した場合は79文字に修正されます。このオプションを指定しなかった場合も79文字と解釈されます。

/Sp<lines> ページ長の設定

1ページの行数を指定します。15から255までの数値で指定します。範囲外の数を設定した場合は63行に修正されます。このオプションを指定しなかった場合も63行と解釈されます。

/St<string> タイトル文字列
/Ss<string> サブタイトル文字列

タイトルメッセージ・サブタイトルメッセージの指定をします。スペースやタブ文字を含める場合には文字列を"でくくってください。

★プリプロセッサオプション

C言語ではコンパイルの前段階として、プリプロセッサを用いて #include, #define, #if, #else, #endif等の解決を行います。
これらの動作を制御するのがプリプロセッサオプションです。

/C コメント削除の停止

プリプロセッサ出力の際にコメントを取り除かなくします。プリプロセッサの出力結果を参照しながらデバッグを行う際に有効です。

/D<name> [=text] マクロの設定

#defineと同様の動作をコンパイル時に行います。

```
cl /DNDEBUG test.c
```

のように指定すると、ソースファイル先頭で #define NDEBUGを行ったものと同義となります。

/E 標準出力へのプリプロセッサ出力
/EP

プリプロセッサの出力を標準出力に出力します。/EPオプションではオリジナルファイルの行番号を示す#line文が出力されません。

/P プリプロセッサ出力ファイルの作成

プリプロセッサの出力結果を .Iの拡張子のファイルとして出力します。

/I <name> インクルードファイルサーチパスの設定

インクルードファイルの検索パスを追加します。通常は環境変数INCLUDEで設定されたディレクトリが検索されますが、/Iオプションで指定することにより、環境変数を変更せずに検索パスを追加することができます。

/X インクルードファイル検索パスの無効化

インクルードファイルの検索パスを無効にします。環境変数INCLUDEの設定を無効にします。

/U<name> 定義済みマクロの削除
/u 定義済みマクロの全削除

指定された名前の定義済みのマクロを消去します。MicrosoftCの場合には次のようなマクロが定義されています。/U<マクロ名>で指定されたマクロを未定義状態にします。/uではすべてのマクロを未定義状態にします。

MSDOS	MS-DOSを使用する
M_I86	8086系のCPUを使用する
M_I86*M	メモリモデルを示す。*にはS,M,L,C,Hのいずれかが入る。NO-EXT-KEYS(拡張機能)を使用する。/Ze/Za参照。

_CHAR_UNSIGNED char型は符号なしである。/J参照。

★言語関係

これらのオプションでは、拡張機能の有無やデバッグオプションの設定等を指定します。

/Za 拡張機能の禁止

MicrosoftC特有の拡張機能を使用不可にします。MicrosoftCではfar,huge,pascal,fortran,interruptといった特有の予約語を持っています。移植性を重視する場合には、それらの予約語は使用しない方がよいでしょう。/Zaオプションはこれらの拡張機能を使用できなくするものです。

/Ze 拡張機能の許可

拡張機能を使用可とします。/Zaオプションを指定しない限りこのオプションは指定されたものと見なされます。

/Zd 行番号情報の付加

シンボリックデバッガ(SYMDEB)で参照する行番号情報を出力します。SYMDEBでソースコードデバッグするためには、/Od /Fmオプションと共に使用しなければなりません。

/Zg プロトタイプ宣言の作成

ソースコード中の関数のプロトタイプ宣言を標準出力に出力します。ヘッダファイルのプロトタイプ宣言を作成するときに有用です。

/Zi ソースコードデバッグ情報の付加

CodeViewデバッガ(CV)でデバッグするための情報を出力するようにします。CVでデバッグを行うためにはこのオプションを指定してコンパイルしなければなりません。

/Zl デフォルトライブラリ情報の削除

デフォルトで参照されるライブラリ名の指定を取り除きます。オブジェクトファイルのサイズを小さくしたい場合に使用します。実行形式のファイルサイズにはまったく影響しません。

/Zp [n] 構造体要素の境界アドレスの設定

構造体の要素をnバイト境界に割り当てます。何も指定されなかった場合、ワード境界に割り当てられます。

/Zpのみ指定された場合、n＝1、バイト境界に割り当てられます。同様の動作をpragmaで局所的に実現することができます。

書式

#pragma pack(1)	バイト境界に割り当てる
#pragma pack(2)	ワード境界に割り当てる
#pragma pack(4)	ダブルワード境界に割り当てる

/Zs 文法チェックのみを行う

構文エラーの検出のみを行います。オブジェクト出力を行わないので、高速に構文チェックを行うことができます。

★浮動小数点

/FPa 簡易ライブラリ呼び出し

簡易数値演算ライブラリを使用します。簡易数値演算ライブラリはサイズが小さく、エミュレートライブラリより高速です。ただし、演算精度は悪く、8087が実装されているシステムでも8087を使用せずに演算を行います。

/FPc エミュレータライブラリ呼び出し

エミュレータライブラリを使用します。8087が実装されている場合には8087を使用して高速に浮動小数点演算を行います。

/FPc87 8087ライブラリ呼び出し

87ライブラリを使用します。8087が実装されていないシステムでは動作しないプログラムとなりま

す。8087が実装されているシステムでは、/FPi87よりも遅く、/FPaよりも速いコードが生成されます。

/FPi エミュレータコード生成

エミュレータ呼び出しをインライン展開します。何も指定しない場合には、このオプションが指定されたものと見なされます。/FPcより高速になります。

/FPi87 8087コード生成

8087コードをインライン展開します。8087が実装されていないシステムでの実行は不可能になります。/FPc87，/FPiより高速に実行できます。

★その他のオプション

/c コンパイルのみを行う

コンパイルのみを行います。通常clコマンドはコンパイルの後リンクを行いますが、このオプションを使用することにより、リンクを行わなくすることができます。makeによってモジュール分割のプログラムを作成する場合に有用です。

/H<number> 外部名の最大長設定

外部定義名の長さを設定します。MicrosoftCでは外部定義名の長さの制限はありませんが、他のコンパイラでは、長い名前の使用がエラーを起こす可能性があります。そのようなコンパイラに移植することを前提とした開発のために用意されています。

/J char型の符号なし設定

char型を符号なし8ビットの数値と見なします。

/Tc<file> 拡張子にかかわらずコンパイルを行う

拡張子が .c以外のファイルをソースファイルとしてコンパイルします。clコマンドでは拡張子が.cの物をコンパイル、拡張子が .cでないファイルをオブジェクトファイルとしてリンクしますが、このオプションにより拡張子が .c以外のものも強制的にコンパイルします。

/V<string> バージョン文字列の埋め込み

オブジェクトファイル中に文字列を書き込みます。オブジェクトファイルの内容を簡単に読むことはできませんが、このオプションによって指定される文字列がそのまま挿入されますので、バージョン管理等に有用です。

/W <number> 警告レベルの設定
/w

警告レベルを設定します。numberには0から3の数値を設定し、数が大きい方がより厳しい検査結果を出力します。/wは /W0と同じ動作になります。/Wで警告レベルを設定しない場合には /W1となります。

/W0
警告を出力しない

/W1
エラーではないが大変危険なコードの場合警告を発します。/W1で出力される警告はエラーと同等に考えてソースを見直した方がよいでしょう。

/W2
返り値が定義されていない関数の使用、void型でない関数でreturn;が用いられた、精度が落ちるデータ変換がなされた場合警告を発します。

/W3
ANSI外の命令、拡張命令、プロトタイプ宣言されていない関数が出現した場合に警告を発します。

2.3.2 LINK

LINKは複数のオブジェクトモジュールを結合してひとつの実行型ファイルを作成します。オブジェクトモジュールは、マイクロソフトが定義したオブジェクトモジュールフォーマット形式のファイルです。このフォーマットはマイクロソフト社が出すコンパイラ(MicrosoftC、MS-FORTRANやMS-PASCAL)やマクロアセンブラ(MASM)の共通フォーマットになっています。またマイクロソフト以外のソフト会社から発売されている言語コンパイラの多くは(ボーランドのTurbo C等)この形式を採用しています。

MicrosoftCでのLINKの役割は、clコマンドが作り出す.OBJファイル同士と、そこで使われているライブラリ関数をライブラリファイルから取り出して結合し、実行ファイルを作成することです。

clコマンド自体では、実行ファイルは作れません。clコマンドはソースプログラムから .OBJファイルを作り出し、その中でLINKを起動しています(/cオプションを付けなかった場合)。

(1) LINKの起動方法

上述のようにLINKは .OBJファイルと .LIBファイルを結合し、.EXEファイルを作成しますので、それらの情報を起動時のパラメータとして渡さなければなりません。

LINKにパラメータを与える方法は3種類あります。

① MS-DOSプロンプトから入力する

```
link [options] objfiles, exefile, mapfile, libfiles
```

optionsにはリンク時のオプションを指定します。optionの詳細については後述します。

objfilesにはオブジェクトファイル名を複数書くことができ、拡張子".OBJ"は省略可能です。複数書く場合は、'+'で区切り列記します。

exefileには作り出す実行型ファイル名を指定します。拡張子".EXE"は省略可能です。何も指定しなかった場合にはひとつ目のオブジェクトファイル名の拡張子が".EXE"に置きかわったファイル名となります。

mapfileにはリンク結果のマップファイル名を指定します。拡張子".MAP"は省略可能です。マップファイルはテキストファイルで、オブジェクトファイルとライブラリを結合した結果の情報が入っています。結合結果の細かい情報を参照したい場合や、シンボリックデバッガ(SYMDEB)を使用してデバッグを行いたい場合などに便利です。何も指定しなかった場合にはマップファイルは作られません。

libfilesには参照するライブラリファイル名を複数書くことができます。MicrosoftCではライブラリ名を特に指定する必要はありません。オブジェクトモジュールにデフォルトライブラリ情報として、参照しなければならないライブラリ名が書き込まれているからです。

② 応答プロンプトにしたがって入力する

MS-DOSプロンプトから"LINK"とのみ入力すると、

```
>link

Object Modules [.OBJ] :
Run File [????????.EXE] :
List File [NUL.MAP] :
Libraries [.LIB] :
```

と、プロンプトが順に表示され、オブジェクトファイル、実行ファイル、マップファイル、ライブラリファイルを1行ずつ入力することができます。[.OBJ]の様に示されているものは、デフォルトの拡張子やファイル名です。変更する必要のない場合には何も入力せずに改行します。拡張子を省略して入力しても構いません。

大きなプログラムになるとオブジェクトファイルを1行に指定しきれなくなる場合があります。そのようなときには、数行にわたってオブジェクトファイルを指定することもできます。

```
>link

Object Modules [.OBJ] : file1+file2+file3+
file4+
```

```
Object Modules [.OBJ] : file5+file6
Run File [FILE1.EXE] :
List File [NUL.MAP] :
Libraries [.LIB] :
```

のように、Object Modulesに入力する行の最後を、'+'で終らせると、再度Object Modulesを尋ねてきますから、必要なだけ入力することができます。

③ レスポンスファイルを使用する。

①で述べた方法でならば、バッチファイルを記述して簡単に起動することができますが、オブジェクトファイルの数が増えてくると1行に書き切れなくなり、多数のオブジェクトファイルをリンクするときには使えません。②で述べた方法ですと、行末を'+'にすることによって多数のオブジェクトファイルをリンクすることができますが、バッチファイルにできないので、起動するたびにいくつもあるオブジェクトファイルを入力しなくてはなりません。

このような問題を解決するためにはレスポンスファイルを使います。レスポンスファイルは、②で述べたような応答プロンプトに対して入力する事柄を1行ずつ書いたファイルです。次のようなレスポンスファイル、LINKFILEがあるとします。

```
/NOI TEST.OBJ +
SUB1.OBJ +
SUB2.OBJ +
SUB3.OBJ
SAMPLE.EXE
SLIBCE.LIB
```

このときに、LINK @LINKFILEのように、レスポンスファイル名の直前に '@'をつけたものをパラメータにしてLINKに渡します。

```
>link @linkfile

Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.
All rights reserved.

Object Modules [.OBJ] : /NOI TEST.OBJ +
Object Modules [.OBJ] : SUB1.OBJ +
Object Modules [.OBJ] : SUB2.OBJ +
Object Modules [.OBJ] : SUB3.OBJ
Run File [TEST.EXE] : SAMPLE.EXE
List File [NUL.MAP] :
Libraries [.LIB] : SLIBCE.LIB
```

このように表示されて、あたかもLINKFILEの内容をキーからタイプしたかのように動作します。

(2) LINKのオプション

CLと同様、LINKにも /HELPオプションが付いています。説明はありませんがオプションの一覧が表示されますので、オプションわからなくなったらLINK /HELPを実行しましょう。

また、LINKはデフォルトのオプションとして環境変数"LINK" の内容を参照します。/NOINORECASEなどの常に使うオプションは環境変数に入れておいた方がよいでしょう。

例: SET LINK=/NOI/B/M

LINKのオプションの指定は、オプション名を特定できるかぎり省略できます。たとえば、"/NODEFAULTLIBRARYSEARCH"オプションは"/NOD"と省略できます。これは、"NOD"で始まるオプションが他にないためです。"/NO"だけですと、"/NOIGNORECASE"などのオプションと区別できませんので"/NOD"まで書く必要があります。

以下のオプション一覧でオプション名の横に付記されたものが省略した書き方です。

/BATCH(/B)

バッチ起動を行います。通常LINKは .OBJや .LIBのファイルが見つからないと、オブジェクトファイルの入っているディスクを入れ換える指示や、ライブラリファイル名の再入力を求めてきます。このオプションを指定すると、そのような場合でも入力要求をせずにエラーを発生して処理を終了します。MAKEの中で使用するのに適しています。

/CODEVIEW(/CO)

コードビューデバッガ(CV)でデバッグするための情報を実行ファイルに書き込みます。CVでソース

コードデバッグをするためには、このオプションとCLコマンドでの/Ziオプションを使用しなければなりません。

/CPARMAXALLOC:number(/CP)

最大アロケートサイズを指定します。numberはアロケートするパラグラフサイズで、1から65535までを指定します。このオプションを使用しない場合、最大アロケートサイズは65535パラグラフとなり、DOSが .EXEファイルをロードする際に最大のメモリブロックを確保しようと試みます。このオプションで最大アロケートサイズを指定すると、適切なメモリを割り当てることができます。しかし、特殊な場合でないかぎり、このオプションを使用することはないでしょう。

/DOSSEG(/DO)

MS-DOS標準のセグメント列にリンクします。MicrosoftCで使用するならば、このオプションを指定する必要はありません。/DOSSEGオプションは次のような順番にセグメントを配置します。

1. クラス名がCODEで終るセグメント
2. DGROUPに属さないその他のセグメント
3. DGROUPのセグメント
   1) クラス名 BEGDATA
   2) クラス名が BEGDATA,BSS,STACK以外
   3) BSS セグメント
   4) STACK セグメント

/DSALLOCATE(/DS)

このオプションは実行ファイル起動時のDSレジスタの初期値をデータセグメントの最上位に置くものです。MicrosoftCではこのオプションを使用することはできません。MASM等でメモリ最上位にロードするプログラムを作成する際に使用するためのものです。

/EXEPACK(/E)

実行ファイルをパックします。同一の数値の連続があった場合にその部分を圧縮します。また、ローディング時間を速くするために再配置テーブルを最適化します。

/EXEPACKを付けてリンクしたプログラムはSYMDEBによってデバッグすることができなくなるので注意が必要です。

EXEPACKユーティリティによって同じ動作を行わせることができます。

/FARCALLTRANSLATION(/FA)
/NOFARCALLTRANSLATION

/FARCALLTRANSLATIONは、FARラベルへのコール命令を、可能ならばNEARコールに置き換えます。呼び出し先のFARラベルのセグメントが呼び出し元のセグメントと同一であれば、アセンブラ言語上での、

```
CALLFAR PROC
```

を、

```
PUSHCS
CALLNEAR PROC
NOP
```

に置換します。これによって実行速度が向上します。しかし、リンカが誤った置換を行う場合もあります。MicrosoftCで使用するならばほぼ問題ないと思われますが、/FARCALLTRANSLATIONオプションを使用するプログラムで不都合が発生した時には、まず/FARCALLTRANSLATIONを行わないでチェックしてみてください。

/NOFARCALLTRANSLATIONは/FARCALLTRANSLATION の指定を無効にします。

/HIGH(/HI)

実行ファイルが読み込まれる際にメモリ最上位に読み込むように指定します。このオプションはMicrosoftCで使用することはできません。

/INFORMATION(/I)

リンクされたオブジェクトファイル名等の情報をエラー出力に出力します。通常の方法では、リダイレクトできませんので、ファイルにしたい場合はERROUTユーティリティを使います。

/LINENUMBERS(/L)

マップファイルに行番号情報を付加します。このオプションはSYMDEBでソースコードデバッグを行う際に必要です。

/MAP:number (/M)

マップファイルを出力します。マップファイルにはグローバルシンボルが名前順、アドレス順にソー

トされた表が入ります。numberはソート可能な最大シンボル数で、指定しなかった場合は2048となります。

/NODEFAULTLIBRARYSEARCH(/NOD)

デフォルトのライブラリ検索を抑止します。MicrosoftCでコンパイルされたオブジェクトファイルにはデフォルトライブラリ情報が含まれています。そのため、ライブラリファイルを指定しなくてもリンクできるのですが、そのライブラリファイルを使用したくない時にこのオプションを付けます。

CLコマンドの/Zlオプションでは、デフォルトライブラリ情報の書き込みを行わない指定をすることができます。

/NOGROUPASSOCIATION(/NOG)

グループ指定を無視します。旧バージョンの互換性のために用意されているオプションで、MicrosoftCで使用することはできません。

/NOIGNORECASE(/NOI)

このオプションを指定しない場合、LINKは外部定義名の大文字と小文字を区別しません。C言語では名前の大文字と小文字をすべて区別するので、MicrosoftCで使用する際は /NOIGNORECASEオプションを必ず使用してください。

/OVERLAYINTERRUPT:number (/O)

オーバーレイ動作のために使用する割込ベクタ番号を変更します。デフォルトでは0x3fが使用されています。numberには0から255までの数値を指定します。

他の割り込みと競合するベクタ番号を指定した場合でもエラーを出しません。使用する際には充分な注意が必要です。

/PACKCODE:number(/PAC)
/NOPACKCODE(/NOP)

/PACKCODEオプションは連続したコードセグメントをグループ化します。numberにはグループ化するコードセグメントの最大サイズを指定します。デフォルトは65530バイトです。/FARCALL-TRANSLATIONと併用することにより、高速な実行ファイルを作成することができます。/NOPACK-CODEは/PACKCODE環境変数LINKによって指定された/PACKCODEオプションを無効にします。

/PAUSE(/PAU)

/PAUSE オプションは実行ファイルを作成する直前にプロンプトを表示して停止します。ディスクベースで作業をする場合に使用します。

/QUICKLIBRARY(/Q)

クイックライブラリを作成します。LINKは通常は .EXEファイルを作りますが、このオプションを指定することにより .QBLファイルを作成します。.QBLファイルについてはQuickCやQuick-BASICに関する書籍を参考にしてください。

/SEGMENTS: number

LINKが取り扱うセグメント数の最大を設定します。デフォルトでは128セグメントまで管理できます。numberには1から3072までの数値を設定します。

/STACK: number

スタックの大きさを設定します。numberには1から65535までの数値を設定します。

2.4 CodeViewの使い方

マイクロソフトコードビューデバッガ(CV)はマウス操作の高機能デバッガです。マイクロソフトから提供されているデバッガは現在のところ3種類あります。

DEBUG: デバッガ
SYMDEB: シンボリックデバッガ
CV: コードビューデバッガ

DEBUGは実行型プログラムの簡単なデバッグ機能を持つもので、オブジェクトレベルでのデバッグだけしかできませんでした。SYMDEBではソースレベルでのデバッグが行えるようになり、数値表現やブレークポイント機能も拡充されました。

MicrosoftCバージョン4.0からはコードビューが付属するようになりました。SYMDEBに比べ、旧コードビューではマウスにより操作ができる、プルダウンメニューによる簡単な操作、ソースコードを表示しながらのレジスタ表示、デバッグ画面と出力画面の切り換え等、高機能なものになりました。

MicrosoftC Ver5.1に付属しているコードビューはさらに機能拡張され、バージョンは2.2になっています。

CV Ver2.2で新規サポートされたのは次のとおりです。

- 80386のサポート
- 8087エミュレータのサポート
- C, BASIC, FORTRAN, Pascal式のサポート
- オーバーレイのサポート
- コマンドの追加

2.4.1 CODEVIEWの起動

起動方法：
cv [options] file [arguments]

CVに続いてオプション、起動ファイル名、起動したファイルに対するオプションや引数を指定します。

★オプション

/B
白黒モードで使用します。

/C<command>
指定されたコマンドを起動時に実行します。

/E
拡張メモリを使用します。

/M
マウスドライバを使用しません。

2.4.2 メニュー内容

(1) Filesメニュー

ファイルをオープンしたり、シェル起動、CVの終了を行うメニューです。マウスで [Files] をクリックするか、GRPHキーに続いて'F'を押すとFilesメニューが表示されます。

[Open...]
ソースファイルをオープンします。

[Dos Shell]
COMMAND.COM を起動します。CVに戻るにはプロンプトから

[Exit]
CVを終了します。

(2) Viewメニュー

画面表示を切り替えるメニューです。マウスで〔View〕をクリックするか、GRPHキーに続いて'V'を押すとViewメニューが表示されます。

[Source]
ソースプログラムを表示します。

[Mixed]
ソースプログラムとオブジェクトの混合リストを

表示します。

[Assembly]
オブジェクトリストを表示します。

[Registers]
画面右側にレジスタの内容を表示するウィンドウを開きます。

[Output]
出力画面を表示します。画面表示を行うプログラムのデバッグに便利です。

(3) Searchメニュー

ソースコード上での文字を正規表現を用いて検索します。マウスで［Search］をクリックするか、GRPHキーに続いて'S'を押すとSearchメニューが表示されます。

[Find]
正規表現を入力し文字列検索を行います。

[Next]
直前に入力された文字列で、順方向に再検索を行います。

[Previous]
直前に入力された文字列で、逆方向に再検索を行います。

[Label]
ラベルまたは関数名を入力して検索します。

(4) Runメニュー

デバッグ中のプログラムの実行をするメニューです。マウスで［Run］をクリックするか、GRPHキーに続いて'R'を押すとRunメニューが表示されます。

[Start]
実行を開始します。

[Restart]
ブレークポイント等で中断したプログラムを最初から実行します。

[Execute]
1ステップごとにトレースしながら実行します。

[Clear Breakpoints]
設定されているブレークポイントをすべて取り消します。

(5) Watchメニュー

ウォッチ式やトレースポイントを設定します。マウスで［Watch］をクリックするか、GRPHキーに続いて'W'を押すとWatchメニューが表示されます。

[Add Watch] (Ctrl+W)
ウォッチ式を追加します。トレース中やブレーク時にこの式の値がウォッチウィンドウに表示されます。

[Watchpoint...]
ウォッチポイントを追加します。式の条件が真になるとプログラムがブレークします。

[Tracepoint...]
トレースポイントを追加します。トレースポイントに設定したメモリの範囲に変更があるとブレークします。実行速度が極端に遅くなるので使用に際しては注意が必要です。

[Delete Watch] (Ctrl+U)
ウォッチポイントを削除します。

[Delete All Watch]
全ウォッチポイントを削除します。

(6) Optionsメニュー

デバッグモードの細かい設定を行うメニューです。マウスで［Options］をクリックするか、GRPHキーに続いて'O'を押すとOptionsメニューが表示されます。

[Flip/Swap]
[Run] メニューの [Execute] や、[Trace] を行った際に毎回実行画面を表示するかどうかを設定します。[Flip/Swap] の左に＞＞が表示されている状態で実行画面の表示を行います。

[Bytes Coded]
[View] メニューで [Mixed] または [Assembly] を選択した際にオブジェクトリストが表示されますが、そのさいにニーモニックに加えて機械語コードも表示するかを設定するオプションです。[Byte Coded] の左に＞＞が表示されている状態で機械語コードの表示を行います。

[Case Sense]
デバッガが大文字と小文字を区別するかを設定します。[CaseSense] の左に＞＞が表示されている状態で大文字小文字の区別を行います。

[386]
80386コードのデバッグを行います。CPUが80386でないシステムで使用することはできません。このオプションを指定すると、[View] / [Registers] で表示させるレジスタ一覧が32ビット表示になります。

(7) Languageメニュー

デバッガで使用する式表記の種類を指定するメニューです。マウスで [Language] をクリックするか、GRPHキーに続いて'L'を押すとLanguageメニューが表示されます。

[Auto]
自動判別を行います。

[Basic]
BASICの式を用います。

[C]
Cの式を用います。

[Fortran]
FORTRANの式を用います。

(8) Callsメニュー

デバッガで使用する式表記の種類を指定するメニューです。マウスで [Calls] をクリックするか、GRPHキーに続いて'R'を押すとCallsメニューが表示されます。

このメニューには決まった内容がありません。現在トレース中の場所の呼び出し関数が引数と共に表示されます。

なお、CodeViewデバッガの操作について、具体的な例を挙げて62ページから説明します。御参照ください。

2.5 MicrosoftC、バージョンによる違い

MicrosoftCのバージョン5.10は、4.0と比べ大幅に機能が向上しています。主な点ではANSI規格案の導入、言語仕様の拡張、ライブラリ関数の新設が挙げられます。

●ANSI規格案の導入

ANSIとはAmerican National Standards Instituteの略で米国の規格制定を行う機関です。この機関では現在、C言語の文法やライブラリ関数についての規格の制定が行われており、MicrosoftCバージョン5.10では、その規格案に準じる形で、言語仕様や関数の変更、拡張が行われました。

●MicrosoftC独自の拡張

従来、MicrosoftCでは記述できなかった割り込みルーチンを実現するinterrupt型がサポートされるようになりました。#pragma命令も追加され、従来のcheck_stackのみではなく、alloc_text, comment, data_seg, function, intrinsic, linesize, loop_opt, message, pack, page, pagesize, same_seg, skip, subtitle, tiltle等のpragmaも使用できるようになりました。

●新しいライブラリ関数

MicrosoftCバージョン5.10では、新たなライブラリ関数が追加されました。graph.hで宣言されるグラフィックライブラリや、_dosで始まるMS-DOS関数、_heap, _nheap, _fheapで始まるヒープデバッグ関数などが追加されました。グラフィックライブラリではPC-9801シリーズやAXシリーズのグラフィック描画を行う関数群をサポートしています。_dosで始まる関数は、MS-DOSレベルでのメモリーのアロケーション、ファイル・デバイス入出力、ディレクトリサーチなどをサポートしています。ヒープデバッグ関数はヒープ処理を行うプログラムのデバッグの際に使用し、ヒープの整合性のチェック（ヒープが破壊されていないか等） ヒープのエントリ取得をサポートしています。

その他、ビットローテート関数等、細かい点での拡張が行われています。

【pragma一覧】

●コード生成に関するpragma

#pragma alloc_text(セグメント名,関数,…)
関数を指定されたセグメントに配置する

#pragma check_stack([{on|off}])
スタックチェックの有無

#pragma data_seg(セグメント名)
データを指定されたセグメントに配置する

#pragma loop_opt([{on|off}])
ループのオプティマイズの有無

#pragma pack([{1|2|4}])
構造体メンバの境界アドレス

#pragma same_seg(変数,…)
変数が同じデータセグメントにあるものと見なす

●インラインに展開するpragma

#pragma function(関数,…)
指定された関数のインライン展開を停止する

#pragma intrinsic(関数,…)
指定された関数をインライン展開する

●リスティングに関するpragma

#pragma linesize(数値)
ソースリスティングの1行の文字数

#pragma page(数値)
改ページする

#pragma pagesize(数値)
1ページあたりの行数

#pragma skip(数値)
改行する

#pragma subtitle(文字列)
サブタイトル文字列

#pragma title(文字列)
タイトル文字列

●その他のpragma

#pragma comment(コメントタイプ[,コメント文字列])
オブジェクトファイルにコメントを埋め込む

#pragma message(文字列)
メッセージを出力する

【CodeViewの使用例】

C言語の文法的な誤りは､clコマンドがエラーやウォーニングとしてメッセージを返してきます。使われている外部関数や変数が無いときには、LINKがエラーを返してくれます。このような場合、メッセージに応じて該当するソースコードを書き直せば、誤りは簡単に直すことができます。

しかし､clコマンドがエラーを返さず､LINKもエラーを起こさないプログラムであるのに、実行時に正常に動作しないようなプログラムは、ソースコードを見直すなりデバッガを用いたりしてバグの発生箇所を突きとめねばなりません。

この項では、MicrosoftC に付属するデバッガCodeViewの使い方を、プログラムを実際にデバッグするステップを追って紹介します。デバッグに用いるプログラムは、ワイルドカードで指定したファイル以外のファイルを削除するプログラムdelex.cです。リストのままですと、引数で与えられたファイルを正常に認識してくれないという症状があります。

CodeViewでデバッグするためには、clコマンドによるコンパイル時に/Ziオプションを用いて、CodeView用デバッグ情報を含む実行形式ファイルを作成するように指定して再コンパイルします。

単一のプログラムの場合ならば/Ziオプションで、cl /Zi delex.c のようなコンパイルを行えばよいのですが、複数のファイルに分かれたプログラムをコンパイルする際には、clでコンパイルのみを行わせ、LINKで改めてリンクを行うのが一般的です。このように、clによるLINKの自動起動を行わない場合、clコマンドの/Ziオプションを付けた上で、LINK時に/CODEVIEWオプションを使用する必要があります。

CodeViewはマウスを使用することができます。そのためにはデバイスドライバとしてmouse.sysが登録されているか、常駐型マウスドライバmouse.comを常駐させておかねばなりません。

clによって作られた、delex.exeをCodeViewでデバッグするには、

```
cv delex
```

と、入力します。以下、実際の画面を追って説明します。

★delex.c★　(CodeView　動作例用プログラム)

```
/* このプログラムは故意にバグを入れてあります。このままでは動作しません */

#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <stdlib.h>

struct filelist {
        int flag;
        char *filename;
        struct filelist *next;
};

void *chkmalloc(size_t size)
{
        void *p;

        if ((p = malloc(size)) == NULL) {
                perror("malloc");
                exit(2);
        }
        return p;
}

char *chkstrdup(char *str)
{
        char *p;
```

```
        if ((p = strdup(str)) == NULL) {
                perror("strdup");
                exit(2);
        }
        return p;
}

struct filelist *getfilelist(int argc, char **argv)
{
        struct filelist root, *p;
        struct find_t fb;
        int r;

        p = &root;
        while (argc-- > 0) {
                for (r = _dos_findfirst(*argv++, _A_NORMAL, &fb);
                     r == 0;
                     r = _dos_findnext(&fb)) {
                        p->next = chkmalloc(sizeof(struct filelist));
                        p = p->next;
                        p->filename = chkstrdup(fb.name);
                        p->next = NULL;
                }
        }
        return root.next;
}

int searchname(struct filelist *list, char *name)
{
        while (list != NULL) {
                if (strcmp(list->filename, name) == 0)
                        return 1;
                list = list->next;
        }
        return 0;
}

int main(int argc, char **argv)
{
        struct filelist *argfiles, *allfiles, *lp;
        char *wildall = "*.*";

        argfiles = getfilelist(--argc, ++argv);
        allfiles = getfilelist(1, &wildall);
        lp = allfiles;
        for (lp = allfiles; lp != NULL; lp = lp->next)
                lp->flag = searchname(argfiles, lp->filename);
        for (lp = allfiles; lp != NULL; lp = lp->next)
                if (!lp->flag) {
                        if (unlink(lp->filename) == -1)
                        perror(lp->filename);
                }
        return 0;
}
```

CodeViewを起動すると、このような画面となります。CodeViewではC言語の他にアセンブラで記述されたプログラムや、CodeView用のデバッグ情報を含まないプログラムのデバッグも行うことができます。ここではデバッグ情報が含まれるC言語プログラムですから、CodeViewはC言語ソースプログラムを画面に表示します。

⇩

ファイル名を取得できないバグですから、その処理を行う関数、getfilelist関数内をトレースします。まずgetfilelist関数の先頭にブレークポイントを設定します。ブレークポイントとは、プログラムの実行が、ある一定のアドレスに達したときに処理を中断するデバッグ機能です。

⇩

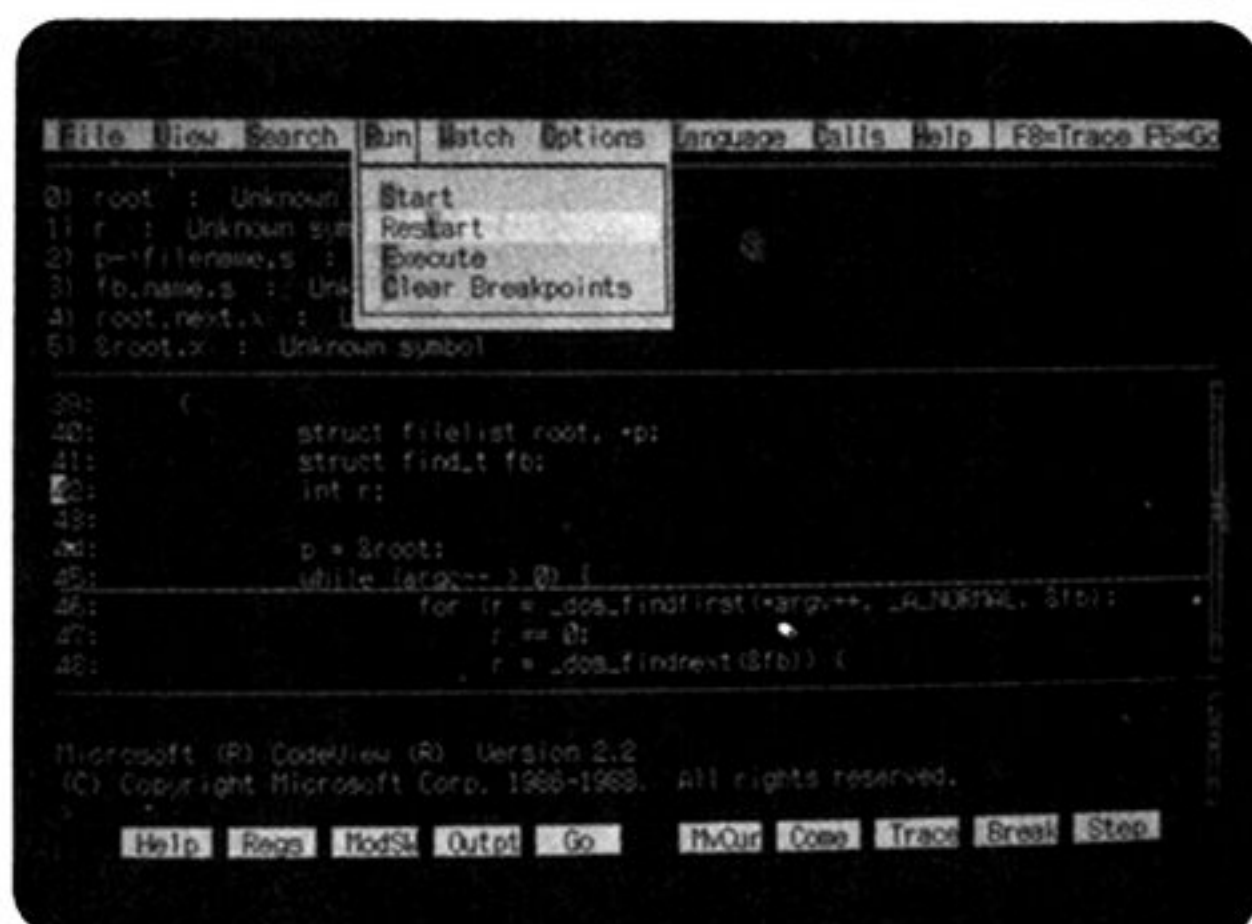
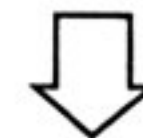
ウォッチの設定を行います。ウォッチは指定された変数やメモリの領域を常時監視する機能です。バグを引き起こす箇所を特定する手がかりとなるような変数を、思いつくかぎりウォッチとして登録します。ここでは、変数root, rや、filelist構造体のポインタpのメンバfilenameの文字列などを表示させています。ウォッチを表示させるためには、^WまたはWatchメニュー内のAdd Watchコマンドを用います。

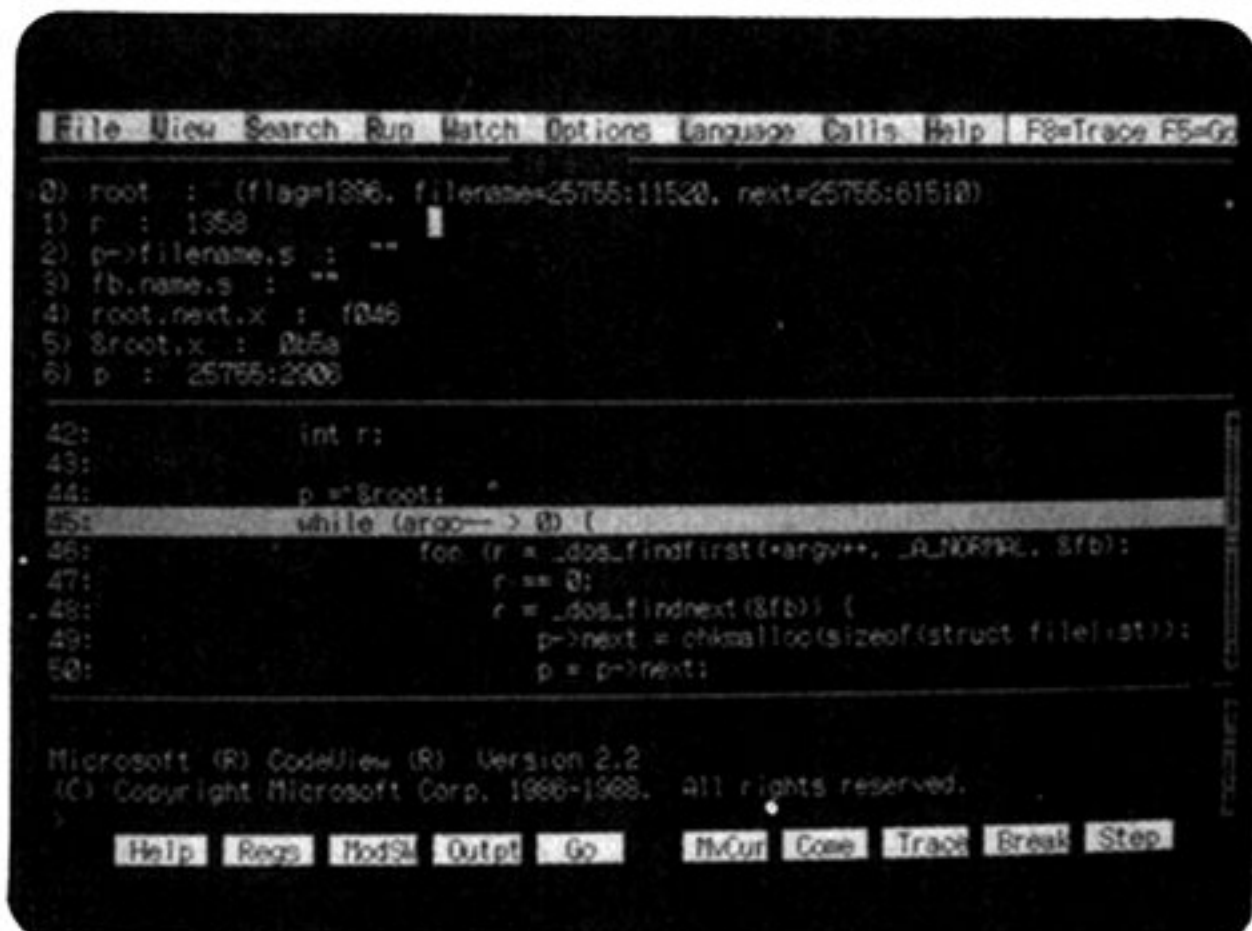
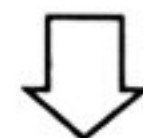
⇩

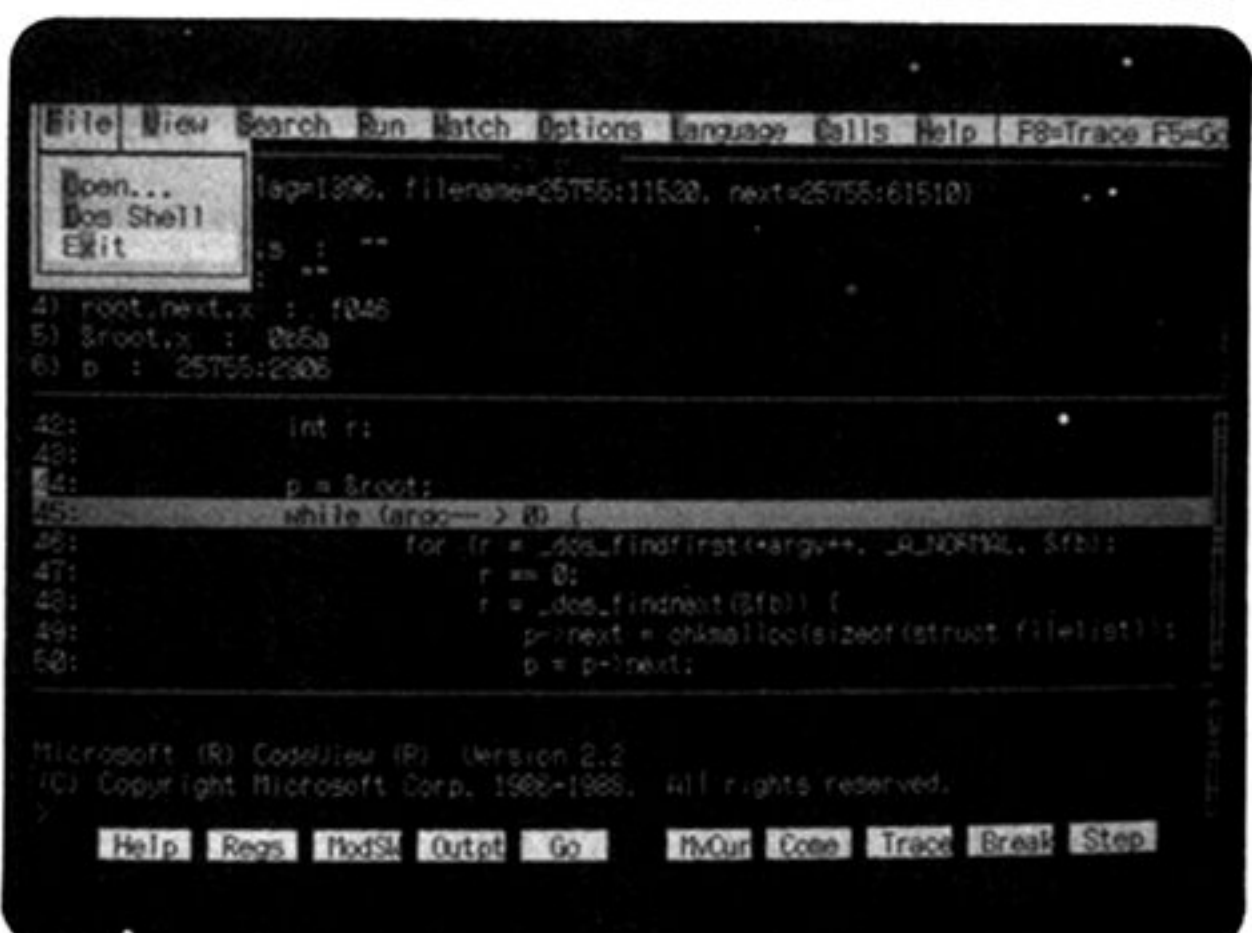
ウォッチを設定したら、各変数がどのように変化するかを監視しながらトレースを行います。気になった変数があったら、順次追加するつもりでトレースを行ってください。

⇩

トレースを順次行うと、ウォッチウィンドウに表示された変数が処理に応じて変化して行きます。変数の変化ひとつでバグの原因が突き止められる場合もあります。

⇩

これで、一通りのトレースが終りました。今までの部分では特に異常は起きなかったので、再度プログラムを実行します。プログラムを再実行するためには、Runメニュー内のRestartを選択します。

⇩

Restartを行うと、先ほど設定したgetfilelist内のブレークポイントでプログラムの実行が停止します。ここで、変数pをウォッチに加えると、ポインタであるpの値が正常な位置を示していないことがわかります。これは変数pが初期化されていないために起こるバグのようです。

⇩

異常箇所がつかめましたので、デバッガを終了します。終了するにはFileメニューのExitを選択します。

第3章　プログラミングのかんどころ

この章では、プログラミング例を通して、実際にはどのようにプログラミングしていくのかを解説します。

C言語についての基礎知識だけでは、実際のプログラミング作業はスムーズには進みません。たとえば、あるエラーが出たときに、どの部分が悪いのかを知るためには、ある程度の経験が必要です。経験が少なければ、正解を見つけられるまで、試行錯誤でかなりの時間を要することになります。また、単独の関数の使い方はリファレンスを参照することでわかるかも知れませんが、その有機的な使い方を身に付けるには、自分でやってみる以外には修得の方法はありません。

MicrsoftCで用意されている関数の中には、似たようなものも多くあります。その実際の違いなどは、単に勉強しただけではなかなかわかりにくいものです。

MicrsoftCでは、あらかじめ用意されている関数に関してはインクルードファイルで定義されています。これら複数のインクルードファイルは、関数の機能ごとに別々のファイルになっています。第2部のリファレンスも、機能分けに基づいた順番で構成されています。

そこでこの章では、サンプルプログラムを通して、その実際の有機的な使い方、すなわち「プログラミングのかんどころ」をご紹介します。すべての関数について網羅してはいませんが、ポイントさえつかめればほとんどの関数の使い方を理解できるようになるでしょう。

3.1 モールス符号フィルタ

ここで紹介するmorse.cは、キーボードから入力されたコードを、モールス・コードに置き換えて、画面に「-•-•」と表示するものです。もちろんモールス・コードとは、無線通信などで使われている電信のコードです。

3.1.1 ストリーム入出力関数の概要

MS-DOSではコンピュータと外部との入出力をファイルと同様に考えています。コンソール、プリンタ、RS-232Cなどは、それぞれCON、PRN、AUXなどとデバイス名が定義されています。その中でもキーボードや画面出力は最もよく使われるもので、これを標準入出力と呼んでいます。通常、標準入力はキーボード、標準出力は画面になっています。

このようなデバイスやファイルとの入出力を行わせるための関数群が、stdio.hで定義されているストリーム入出力関数です。

ストリーム名は次のように定義されています。

stdin	標準入力(通常はキーボード)
stdout	標準出力(通常はスクリーン)
stderr	標準エラー出力(通常はスクリーン)
stdaux	標準補助入出力(通常はRS-232C)
stdprn	標準プリンタ出力

これらのストリーム以外の入出力デバイスを使用するためには、ファイルと同様にfopen関数を使ってオープンしないといけません。使用した後には、fclose関数を使ってクローズします。ストリーム入出力関数では、オープンしたときに返されるFILE構造体へのポインタを使ってアクセスします。

ストリーム出力をするためには、次のような関数を使います。

fprintf	書式付き出力
fputc	1byte出力
fputchar	標準出力への1byte出力
fputs	1行出力
fwrite	固定長出力

また、ストリームからの入力をするためには、次のような関数を使います。

fscanf	書式付き入力
fgetc	1byte入力
fgetchar	標準入力からの1byte入力
fgets	1行入力
fread	固定長入力

これら入力と出力関数は、機能的に考えると順に1対1対応しています。また、これらの関数の他にも、putc、getsなどのようにfのない標準入出力専用の関数群も用意されています。

ストリーム入出力は、バッファリングされます。C/Rコードを出力したときと、ファイルを正常にクローズできたときに、バッファの内容はフラッシュ、すなわち出力されます。ただし、stderr、stdauxはバッファリングされません。バッファリングをサポートされていないこれらのストリームにバッファを割り付けたり、その反対に、バッファを解除するための関数として、setbuf、setvbuf関数があります。また、fflushやflushall関数を使って、強制的にバッファをフラッシュすることもできます。

ストリーム関数を使う上で、リダイレクト機能とパイプ機能は忘れてはならないことです。リダイレクトとは、標準入出力を別のストリームに割り当てることができるものです。たとえば、TYPEコマンドはテキストファイルの内容を標準出力に表示するものですが、

```
A>type readme.doc >prn
```

とすると、出力は標準プリンタ出力に制御が移ります。つまり、標準入出力だけを使ったプログラムを書いても、プリンタやその他のストリーム、ファイルなどと入出力が可能なのです。したがって、フィルタなどのプログラムは、標準入出力を使ったもので他のストリームにも対応できるということです。

●ストリーム入出力関数の使い方

morse.cにおいて、モールス・コードの内容は文字列の配列に取ってあります。入力されたキャラクタコードから、スペースのキャラクタコードを減算することによって、配列の添字を得ています。あとは、配列の内容を標準出力に出力しています。和文と欧文によって、実際のコードの前にシフトする符号を出力していますので、若干処理が複雑になっています。このようなフィルタ・プログラムは、キャラクタコードの並びを利用することによって、能率良く配列にアクセスすることができます。

このプログラムは、リダイレクトやパイプを使用することによって、いろいろな使い方ができます。たとえば、

```
A>morse <test.doc >prn
```

とすれば、test.docの内容をモールス符号化したものをプリンタに出力できます。また、

```
A>dir | morse
```

とすれば、ディレクトリの内容はモールス符号化されて出力されます。

必ずファイルを作らなければならないとき以外には、このように標準入出力を使って書いたプログラムの方が汎用性があります。

プログラム1　★morse.c★

```
/*********************************************

               Morse Filter Program

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include    <stdio.h>
#include    <dos.h>
#include    <string.h>


#define MAX      256
#define TRUE     -1
#define FALSE     0

void main();
int bell(int);
void usage();


char    *mcode[MAX] = { " ", " ", ".-..-.", " ", " ", " ", ". ...", ".----.",
                        "-.--.", "-.--.-", "-..-", ".-.-.", "--..--", "-....-",
                        ".-.-.-", "-..-.", "-----", ".----", "..---", "...--",
                        "....-", ".....", "-....", "--...", "---..", "----.",
                        "---...", "---...", " ", "-....-", " ", "..--..", " ",
                        ".-", "-...", "-.-.", "-..", ".", "..-.", "--.",
                        "....", "..", ".---", "-.-", ".-..", "--", "-.", "---",
                        ".--.", "--.-", ".-.", "...", "-", "..-", "...-",
                        ".--", "-..-", "-.--", "--..", " ", " ", " ", " ", " ",
                        " ", ".-.-.-", "-.--.-", ".-..-.", "--..--", "---...",
                        ".---", "--.--", ".-", "..-", "-.---", ".-...", ".--",
                        "-..--", "--", ".--.", ".--.-", "--.--", ".-", "..-",
                        "-.---", ".-...", ".-..", "-.-..", "...-", "-.--",
                        "----", "-.-.-", "--.-.", "---.-", ".---.", "---.",
                        "-.", "..-.", ".--.", ".-.--", "..-..", ".-.", "-.-.",
                        "....", "--.-", "..--", "-...", "--..-", "--..", ".",
                        "-..", "-..-", "..-.-", "-", "..-.-", "-..-.", ".--",
                        "-..--", "--", "...", "--.", "-.--.", "---", ".-.-",
                        "-.-", ".-.-.", "..", "..--."
        };

char    *jcode = "-..--- ";
char    *acode = "...-. ";

int     w_flag = 0;
int     wabun = 0;

void main(int argc, char **argv)
{
    unsigned long   w;
    int             i, j, k, keyin[128], c;
    w = (long)(0x0fff - (40 * 0x51));
    j = 0;

    if (argc > 2) {
        usage();
        exit(2);
    } else if (argc > 1) {
        i = abs(atoi(argv[1]));
        if (0 < i && i < 50)
            w = (long)(0x0fff - (i * 0x51));
        else {
```

```
            usage();
            exit(2);
        }
    }

    while((c = getchar()) != EOF) {
        if (c == '¥n')
            putchar('¥n');
        if (0x60 < c && c < 0x7b)
            c = toupper(c);
        if (c < 0x20 || (0x5f < c && c < 0xa1) || 0xdf < c)
            continue;
        bell(c);
    }
    exit(0);
}


int bell(int i)
{
    char    tmp_code[20], dum_code[20];
    int     j, k;

    if (0x5f < i) {
        wabun = 1;
        strcpy(tmp_code, mcode[i - 0x60]);
    } else {
        if (0x1f < i && i < 0x40 && wabun == 1)
            wabun = 1;
        else
            wabun = 0;
        strcpy(tmp_code, mcode[i - ' ']);
    }

    if (w_flag != wabun) {
        w_flag = wabun;
        strcpy(dum_code, tmp_code);
        if (w_flag == 0)
            strcpy(tmp_code, acode);
        else
            strcpy(tmp_code, jcode);
        strcat(tmp_code, dum_code);
    }

    for (j = 0; j < strlen(tmp_code); j++)
        putchar((int )tmp_code[j]);
    putchar(' ');
    return(TRUE);
}


void usage()
{
    puts("Morse Touch Voicer by Naoshi Kimura");
    puts("Unknown Optinon");
    puts("usage : morse <inputfile >outputfile");
}
```

3.2 人工無脳(会話)プログラム

ここでは、低水準入出力関数、文字列操作関数、日本語文字列操作関数などを使用したプログラムとして、人工無脳プログラムrie.cと、そのファイルメンテナンスプログラムriemnt.cを紹介します。

人工無脳は、人工知能のパロディのプログラムで、人間と会話しているようにキーボードとスクリーンを通じてコンピュータと会話するものです。最初はまったくデータが無いので会話になりませんが、データが増えていくにしたがって、それなりに会話できるようになります。このプログラムでは人間の入力した文章の一番長い単語をキーとして用い、それに対応する解答データを持っています。知らない言葉のときには、その次に入力された文章を解答データとしてファイルに書込みます。すなわち学習するわけです。

3.2.1 低水準入出力関数の概要

低水準入出力関数はio.hで宣言されています。

ストリーム入出力関数が、バッファリングされるのに対して、低水準入出力関数は、バッファリングされません。また、フォーマット化もされません。ストリーム入出力関数では、FILE構造体を使ってアクセスしましたが、低水準入出力関数はMS-DOSがファイルにアクセスするときに使用するファイルハンドルを表わす整数値でアクセスします。すなわち、よりオペレーティング・システムに近いものです。

低水準入出力関数では、ストリームでも使われていた以下のデバイスについては、あらかじめオープンされているので、あらためてオープンする必要はありません。ただし、ファイルハンドルは次のように決められているので、そのファイルハンドルを使用することになります。

ストリーム	ハンドル	デバイス
stdin	0	標準入力 (通常はキーボード)
stdout	1	標準出力 (通常はスクリーン)
stderr	2	標準エラー出力 (通常はスクリーン)
stdaux	3	標準補助入出力 (通常はRS-232C)
stdprn	4	標準プリンタ出力

これ以外のファイルにアクセスするときには、あらかじめopen関数を使って、ファイルをオープンしなければなりません。終了時にはclose関数でファイルをクローズしなくてはいけません。

ファイルへの出力関数としてはwrite関数、入力関数としてはread関数を使用します。現在のファイルの位置はtell関数によって知ることができます。読み書きをしなくても、ファイル中の好きな位置にlseek関数を使って移動することもできます。

ファイルとの入出力はバッファリングされませんから、自分でバッファを用意します。したがって、低水準入出力関数は固定長の住所録などのようなファイルにアクセスするプログラムに便利です。

3.2.2 文字列操作関数・日本語文字列操作関数の概要

文字列操作関数はstring.h、日本語文字列操作関数は、jstring.hで宣言されています。

C言語においては、文字列はchar型の配列として表されます。したがって、配列の1byteが文字1byteに相当します。

また、文字列の終了はヌル文字'¥0'で表わされます。gets関数などで文字列を読み込んだ場合にはヌル文字が付加されますが、ヌル文字が無い場合、文字列操作関数は予想外の動きをします。printf関数などでも、文字列がヌル文字で終わっていないときには正常に表示できません。

strcpy、strcatなど、文字列を別の領域にコピーするときには、受け側の領域を確保するのはプログラマの責任です。また、ヌル文字も1byteとして扱われますから注意が必要です。確保したつもりが、元の文字列がヌル文字で終了(ターミネート)していないときには、その後ろの領域も含んでヌル文字が出現

するまでの文字列を受け側にコピーしてしまい、他の領域を破壊してしまいます。文字列操作をするときに問題となるのは、このように他の領域を破壊する可能性です。

コピーする文字列の長さが決まっているような場合には、 memではじまる関数(memory.hで宣言されているバッファ操作関数)を使用すれば、ヌル文字で終了していない文字列も扱うことができます。

日本語文字列操作関数は2bytes文字を処理するときに使われます。1byte文字と混合していても使用可能です。ただし、文字列操作関数と違って、コピーなどをするときの単位は、byteではなく文字数です。したがって、バッファサイズなどを設定するときにも注意が要ります。日本語文字列操作関数は文字列操作関数と似ているようで、実際には異なるものですから、相応に注意しないと思った通りの動きをしてくれません。

文字列操作関数でバッファをコピーする関数は、strcpy「送り先のポインタ、元のポインタ」といったように、転送先を先に書きます。文字列の比較では、どちらでも関係ありません。指定文字数コピーならば、strncpy「送り先のポインタ、元のポインタ、bytes数」というように、関数名に「n」が付き、コピーすべきbytes数を指定します。これは文字列比較でも同じです。また、strncmpiのように「i」が付くと、大文字小文字の区別をしません。文字列操作関数には実に多くの関数がありますが、このような法則性を理解すれば憶えることも難しくはないでしょう。

●低水準ファイル入出力の使い方

rie.cではキーワードと解答文からなる構造体munouをファイル入出力のバッファとして持っています。

ファイルは、リード・ライトモードでオープンされ、無いときにはクリエートされます。このときの定数は、fcntl.h、sys¥stat.hで定義されています。ファイルはeofまで読み込まれ、キーワードだけがchar型2次元配列kwywordにコピーされます。

ファイルは固定長で、1レコードはsizeof(MUNOU)だけの大きさを持ちます。したがって、キーワードを発見したときには、lseek関数を使ってその配列の添字×sizeof(MUNOU)だけファイルの先頭からポインタを移動すれば、そのレコードにたどり着くことができるわけです。そこでread関数によって1レコード分をバッファに読み込むことができます。書き込むときにも同様です。

メンテナンスプログラムriemnt.cでは、rie.cと同様の関数を使用してキーワードによるサーチと1レコードずつの移動、トップレコード・ボトムレコードへの移動を実現しています。また、解答文だけではなくキーワードも書き換えることができるようになっています。

デリートはキーワードの先頭を、デリートフラグ「9」にすることで実現しています。固定長のファイルのときには、デリートされたレコードの処理が問題になります。フラグを立てておくだけでは、ディスクのスペースの無駄になってしまいます。しかし、そこを詰めるためには一度別のファイル名ファイルを作成し、デリートされていないレコードのデータだけを書き、その後リネームするという手法を使わないといけません。ここでは、デリートした後で、「9」をサーチして内容を書き変えればよいということにして、この処理は行っていません。

このプログラムは構造体の項目をもっと増やし、それだけの内容を表示・入力できるようにすれば、すぐに住所録などに改造できます。また、キーワードを別の項目についても設定すれば、名前と住所の都道府県名などの複数の条件によって検索することも可能でしょう。

●文字列操作関数・日本語文字列操作関数の使い方

文字列操作は、先ほどの注意を意識していれば、それほど難しいことはありません。

ここで、文字列操作関数を使う上でのポインタについてお話ししましょう。

この例では使っていませんが、

```
char src [20] , dst [40] ;
```

などと宣言されている文字列srcとdstがあったとき、文字列srcの3文字目から後を、文字列dstにコピーすることを考えてみましょう。配列は0から始まっていますから、3文字目はsrc [2] になります。しかし、

```
strcpy(dst, src [2] ) ;
```

では、いけません。正解は、

```
strcpy(dst, &src [2] ) ;
```

です。strcpyの引数はポインタですが、src [2] ではその内容(実体)を表してしまうのです。ただsrcだけの場合には、srcはその配列の先頭アドレスを示しますから、&srcなどとする必要はないのです。もしも&を付けたいのであれば、&src [0] としなくてはいけません。

また、宣言において、

```
char *delm = "  !?#$%&'()=-,.、。 ・_ [] 「」
|{}";
```

と、初期化することができます。このときには、*delmと、「*」が付いていることに注意してください。この宣言の後、ただdelmとすると上文字列定数のあるポインタを指します。したがって、

```
char delm [20] ;
```

などと書いた場合と同様に、文字列操作関数では

```
strcpy(dst, delm);
```

などと、表わすことができるわけです。

ところで、文字列の配列を扱おうとするときには、文字列自体が配列ですから、1次元配列ならば2次元配列として宣言しないといけません。このとき、文字列自体の配列と、その並びの配列、ふたつの添字が必要になりますが、文字列の長さを表わす方の配列添字は後ろにもって来ないと、文字列は連続したアドレスになりません。

人工無脳プログラムのmainで、データファイル名を指定している部分を見てください。strcpy関数などは、戻り値自体がdstを示すポインタになりますから、

```
strcat(strcpy(fname, mname), ".dat");
```

とすれば、mnameの内容をfnameにコピーして、さらにその後ろに「.dat」を追加できます。このようにすれば、1行だけで複数の処理が可能です。

人工無脳プログラムのword_detect関数中では、文章を単語に分解するときに、jstrtok関数を使用しています。strtok関数やjstrtok関数は、第1パラメータの文字列中に、第2パラメータの文字列(デリミタ文字列)の要素が含まれているかを調べ、見つけた最初のポインタを返す関数です。この関数で、次の要素を探すときには、第1パラメータに「NULL」を指定すれば、順次次の要素のポインタを返してくれます。このプログラムでは、デリミタ文字列を発見して単語に分解しつつ、単語の文字列の長さを計算して、一番長い単語を捜し出しています。

また、こうして見つけたキーワードと、キーワード配列の中から一致するものを捜し出す関数find_wordでは、部分一致で捜し出しています。最初はキーワードの文字列分の長さで一致を探します。無いときには、後ろ1bytesを削って、さらに部分一致を捜しています。ここでは、strncmp関数の長さの指定を変えて調べています。

●環境変数の取得

また、このプログラムではgetenv関数を使い、人工無脳の名前を環境変数「MUNOU」を参照して得ています。無いときには、デフォルトの「Rie」になります。getenv関数は指定された環境変数へのポインタを返しますから、mname=getenv("MUNOU")のように、ポインタを代入することで以後、mnameで参照できるようになります。strcpy関数のように内容をコピーしたのではないところに注意してください。

●コマンドラインからの引数の取得

自分の名前については、このプログラムの起動時に

```
A>rie naoshi
```

のように、引数(ひきすう)として与えます。

mainの括弧の中にあるargcの位置には、プログラムのあるファイルのパス名を含めた引数の数が整数値として入ります。したがって、今の例では2が入ります。argcが1であれば、引数は無いということです。

argvは、同様にその内容を示すポインタを示します。rie.exeがb:¥binのディレクトリにあるとすれ

ば、argv [0] は、"b:¥bin¥rie.exe"、argv [2] は、"naoshi"というchar型のポインタを示します。(argv [0] は、MS-DOS3.1以降ではパス名、2.11では、ファイル名が入ります。)

　コマンドラインからの引数によって、処理のしかたを変える例として、arg.cを示します。ここでは示されていないusage関数には、使い方を標準出力するプログラムを書きます。このように、引数を使うことによって、プログラムを起動してからいろいろな選択をしなくても処理の流れを変えることができます。

【rieの使い方】

会話には全角を使用してください。文章は{,.、。□(空白)}などで区切って入力してください。半角では期待した動作は保証されません。

環境変数「MUNOU」によって、お話できる相手を選ぶことができます。指定の無いときには「Rie」になります。

また、defファイルによって、知らないときの反応などの文章を書き変えることができます。

コマンドラインからのパラメータは、自分のハンドルネームになります。指定のないときには「You」になります。

学習しますから、どんどん言葉を覚えます。知らない言葉を言ったときには、一番長い単語をキーワードとして覚えます。次にあなたが打った文章が、その答えの文章として登録されます。次から、キーワードが出てくるとあなたが教えた文章を話します。

```
>set MUNOU=Rika <CR>
>rie ユカリン <CR>
```

ならば、無脳のハンドルネームは［Rika］。あなたのハンドルネームは［ユカリン］と表示されます。defファイルは普通のエディタで編集できます。全部で9行からなり、

1：　はじまりの挨拶 1
2：　はじまりの挨拶 2
3：　から打ちしたときの反応
4：　おしまいのキーワード
5：　おしまいのメッセージ
6：　短い文しか打たなかったときの反応
7：　単語の切れ目のわからない､長ったらしい文を打ったときの反応
8：　知らないときの言葉・前半
9：　知らないときの言葉・後半

と、なっています。8行目と9行目の間には、プログラムで無脳が知らなかったキーワードが入ります。

サンプル・rika.def

```
-----------------------------------------------
ハロー、私は人工無脳のＲＩＫＡです。
私と、お話をやめたいときは、「さよならＲＩＫＡ」
　　と打ってね。
からうちしちゃ、いやぁ！
さよならＲＩＫＡ
ばいばい。また、あそんでね。
ちゃんと、お話してね。
えっと、単語の間は、ちゃんとあけてね。
うーん、
って、知らない言葉だわ。
-----------------------------------------------
```

学習結果は随時､rika.datに書込まれます。間違って変な学習をしたときには、riemnt.exeで修正が可能です。プログラムを終了したいときには、おしまいのキーワードを入力してください。

【riemntの使い方】

riemnt.exeは、rie.exeと共通フォーマットのデータファイルをメンテナンスするプログラムです。環境変数とコマンドラインのパラメータについては、rie.exeと同様に設定できます。defファイルについてはサポートしていません。

rieのデータは、キーワードと解答文が1対1に対応した型になっています。

riemntの起動時はコマンドモードです。このモードでは、次のような選択が可能です。

h: Lower Rec	ひとつ前のレコードを表示
l: Upper Rec	ひとつ後のレコードを表示
j: Bottom Rec	最終レコードを表示
k: Top Rec	最初のレコードを表示
s: Search KeyWord	キーワードによるサーチ
m: Modify Rec	カレントレコードの修正
d: Delete Rec	カレントレコードのデリート
q: Quit	プログラムの終了
?: Help	ヘルプメッセージの出力

★hjklコマンド

HJKLで、レコードをサーチすると、

```
>h
[0000] key : はろ
  answer   : わたしは、無脳のゆかりよ。
```

のように、キーワードと、その解答文章が表示されます。もし、トップレコードかボトムレコードですと、ベルが鳴って表示されません。

★sコマンド

「s」のキーワードサーチを選ぶと、あなたのハンドルネームが表示されますから、キーワードを入力してください。今、「もしもし」というキーワードを発見したとすると、

[Rie] はいはい %
[0089] key : もしもし
answer : はいはい

と無脳が答え、さらにキーワードと解答文を表示します。

キーワードが無かったときには、

Not found !
KeyWord : もしもし
answer :

と表示され、入力待ちになりますから、「はいはい」などと答えれば、

answer : はいはい <CR>
OK ? (n/else) :

と、訊いてきます。ここで「n」を押せば、登録されずにコマンドモードに戻ります。「n」以外のキーを押せば、登録され、

OK ? (n/else) : Completed !
[0089] key : もしもし
answer : はいはい

と表示されます。

★mコマンド

「m」では、カレントレコードのキーワードと解答文を修正できます。カレントレコードは、hjklコマンドや、sコマンドで移動した、現在のレコードを指します。

カレントレコードの内容が、

[0000] key : はろ
answer : わたしは、無脳のゆかりよ。

であるときに、「m」コマンドを実行すると、

>m
Enter New Words and Answer !
key :

と表示されますから、現在のキーワードを変更したいときには、新しいキーワードを入力して下さい。ただし、既に登録してあるキーワードでも二重登録されてしまいますから注意してください。二重登録されたときには、小さい方のレコードが解答文となります。キーワードを変更する必要がないときにはリターンだけを押してください。

key : <CR>
answer :

と、表示されますから、ここでは「はろー、私は無脳のりえよ。」と直してみましょう。

answer : はろー、私は無脳のりえよ。 <CR>
OK ? (n/else) :

と、訊いてきます。ここで「n」を押すと変更されません。「n」以外のキーを押すと、

OK ? (n/else) : Completed !
[0000] key : はろ
answer : はろー、私は無脳のりえよ。

と、変更された内容が表示されます。

★dコマンド

これは、mコマンドとほとんど同様の操作でOKです。ただし、レコードのキーワードが、自動的に「9___________」にされるだけです。完全にデリートされません。

デリートされた(ことになっている)レコードはsコマンドでサーチできますから、mコマンドで書き換えることによって、有効にレコードを使用できるでしょう。

★qコマンド

「q」で、すべての処理を終了します。

>q
[Rie] ばいばい。また、メンテしてね。 %

で、おしまいです。

無脳は、辞書が命なので、こまめにメンテしてやってください。

```
/*******************************************

          人工無脳  りえ   Ver.1.0

   Copyright (c) Naoshi Kimura   1988

*******************************************/

#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <jstring.h>
#include        <io.h>
#include        <fcntl.h>
#include        <sys¥types.h>
#include        <sys¥stat.h>

#define TRUE    -1
#define FALSE   0

#define KEYLEN  13
#define ANSLEN  80

typedef struct {
        char    key_word[KEYLEN];
        char    answer[ANSLEN];
}       MUNOU;

MUNOU   munou;

char    keyword[1024][KEYLEN], words[KEYLEN + 20], *mname, fname[15];
char    def_words[10][80], dname[15];
int             maxrec;

int             def_open(void);
int             dic_open(void);
int             dic_close(int);
void    rie_says(char *);
int             put_answer(int , int);
static char     *get_answer(int , int);
int             find_word(int , char *);
static int      word_detect(char *);


int             def_open(void)
{
        FILE    *fp;
        char    *c;
        int             i;

        if ((fp = fopen(dname, "r")) == NULL) {
                strcpy(def_words[0], "はろー、私は人工無脳のりえです。");
                strcpy(def_words[1], "私と、お話をやめたいときは、「さよなら  りえ」
と打ってね。");
                strcpy(def_words[2], "ＤＤＸちゃないけど、からうちしないでね。");
                strcpy(def_words[3], "さよなら  りえ");
                strcpy(def_words[4], "ばいばい。また、遊びましょうね。");
                strcpy(def_words[5], "もう少し、ちゃんとお話ししてよ！！");
                strcpy(def_words[6], "単語の間は、ちゃんとあけてね。");
                strcpy(def_words[7], "すみませんが、");
                strcpy(def_words[8], "って、どういうことでしょう？");
        } else {
                for (i = 0; i < 9; i++) {
                        if (fgets(def_words[i], 80, fp) == NULL) {
                                printf("illegal format %s !¥n", dname);
                                fclose(fp);
                                abort();
                        }
```

```
                }
        }
        for (i = 0; i < 9; i++) {
                if ((c = strchr(def_words[i], '¥n')) != NULL)
                        *c = '¥0';
        }
        fclose(fp);
        return(TRUE);
}


int             dic_open(void)
{
        int             fh, i;

        if ((fh = open(fname, O_RDWR | O_CREAT, S_IREAD | S_IWRITE)) == -1) {
                perror(fname);
                abort();
        }

        i = -1;
        while(!eof(fh)) {
                if (read(fh, (char *)&munou, (unsigned int)sizeof(MUNOU)) == -1)
 {
                        perror(fname);
                        close(fh);
                        abort();
                }
                strncpy(keyword[++i], munou.key_word, KEYLEN - 1);
                keyword[i][KEYLEN - 1] = '¥0';
        }
        maxrec = i;
        return(fh);
}


int             dic_close(int fd)
{
        if (close(fd) == -1) {
                perror(fname);
                abort();
        }
        return(TRUE);
}


void    rie_says(char *bun)
{
        printf("[%s] %s  %%¥n", mname, bun);
}


int             put_answer(int fn, int rec)
{
        if (lseek(fn, (long)(sizeof(MUNOU) * rec), 0) == -1) {
                perror(fname);
                puts("rie.dat / lseek");
                return(FALSE);
        }
        if (write(fn, (char *)&munou, (unsigned int)sizeof(MUNOU)) == -1) {
                perror(fname);
                puts("rie.dat / write");
                return(FALSE);
        }
        return(TRUE);
```

```
}


static char     *get_answer(int fn, int rec)
{
        static char *error = "error";

        if (lseek(fn, (long)(sizeof(MUNOU) * rec), 0) == -1) {
                perror(fname);
                return(error);
        }
        if (read(fn, (char *)&munou, (unsigned int)sizeof(MUNOU)) == -1) {
                perror(fname);
                return(error);
        }
        return(munou.answer);
}


int             find_word(int fn, char *dumkey)
{
        int             i, j, k;

        k = strlen(dumkey);
        strncat(dumkey, "                    ", KEYLEN - k);
        for (i = KEYLEN - 1; i > KEYLEN - 8; i--) {
                for (j = 0; j <= maxrec; j++) {
                        if (strncmp(keyword[j], dumkey, i) == 0) {
                                rie_says(get_answer(fn, j));
                                return(TRUE);
                        }
                }
        }
        return(FALSE);
}


static int      word_detect(char *bun)
{
        char    dwords[10][KEYLEN + 20], *dtkn;
        char    *delm = "　！？＃＄％＆’（）＝－，．、。・＿［］「」｜｛｝";
        int             i, k, l1, l2, kn;
        int             d_len[10];

        k=0;
        dtkn = jstrtok(bun, delm);
        while(dtkn != NULL) {
                strncpy(dwords[k], dtkn, KEYLEN + 19);
                dwords[k][KEYLEN + 19] = '¥0';
                d_len[k] = strlen(dwords[k]);
                k++;
                dtkn = jstrtok(NULL, delm);
        }
        kn = k;
        l1 = 0;
        l2 = 0;
        for(i = 0; i < k - 1; i++) {
                if(d_len[l1] < d_len[i+1]) {
                        l2 = l1;
                        l1 = i+1;
                }
        }
        strncpy(words, dwords[l1], KEYLEN + 17);
        strncat(words, "                                        ", KEYLEN - strlen(dword
s[l1]) + 19);
        words[KEYLEN + 19] = '¥0';
        return(kn);
}
```

```
void    main(int argc, char **argv)
{
        int     fn, newword, kn;
        char    bun[128], name[40], unknown[100], *c;

        if (argc == 1)
                strcpy(name, "You");
        else
                strcpy(name, argv[1]);

        if ((mname = getenv("MUNOU")) == NULL)
                mname = "Rie";
        strcat(strcpy(fname, mname), ".dat");
        strcat(strcpy(dname, mname), ".def");

        def_open();
        fn = dic_open();
        newword = 0;
        rie_says(def_words[0]);
        rie_says(def_words[1]);

        while(1) {
                printf("[%s] ", name);
                gets(bun);

                if (bun[0] == '¥0') {
                        rie_says(def_words[2]);
                        continue;
                }

                if (strcmp(bun, def_words[3]) == 0) {
                        rie_says(def_words[4]);
                        break;
                }

                if (jstrlen(bun) < 2) {
                        rie_says(def_words[5]);
                        continue;
                }

                if (newword == 1) {
                        strncpy(munou.answer, bun, ANSLEN - 1);
                        munou.answer[ANSLEN - 1] = '¥0';
                        put_answer(fn, maxrec);
                        newword = 0;
                }

                kn = word_detect(bun);
                if (strstr(words, "  ") - words > 25) {
                        rie_says(def_words[6]);
                        continue;
                }

                if (find_word(fn, words) == FALSE) {
                        newword = 1;
                        strncpy(munou.key_word, words, KEYLEN - 1);
                        strncpy(keyword[++maxrec],  words, KEYLEN - 1);
                        if ((c = strstr(words, "  ")) != NULL)
                                *c = '¥0';
                        strcat(strcat(strcpy(unknown, def_words[7]), words), def
_words[8]);
                        rie_says(unknown);
                }
        }
        dic_close(fn);
}
```

プログラム3　★riemnt.c★

```
/*********************************************

      人工無脳りえ　メンテナンス Ver. 1.0

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <jstring.h>
#include        <conio.h>
#include        <io.h>
#include        <fcntl.h>
#include        <sys¥types.h>
#include        <sys¥stat.h>

#define TRUE    -1
#define FALSE   -2

#define KEYLEN  13
#define ANSLEN  80

typedef struct {
        char    key_word[KEYLEN];
        char    answer[ANSLEN];
}       MUNOU;

MUNOU   munou;

char    keyword[1024][KEYLEN], words[KEYLEN + 20], *mname, fname[15];
int             maxrec;

int             dic_open(void);
int             dic_close(int);
void    rie_says(char *);
int             put_answer(int, int);
static char     *get_answer(int, int);
int             find_word(int, char *);
static int      word_detect(char *);
int             search_word(int, char *);
void    modify_rec(int, int);
void    delete_rec(int, int);


int             dic_open(void)
{
        int             fh, i;

        if ((fh = open(fname, O_RDWR | O_CREAT, S_IREAD | S_IWRITE)) == -1) {
                perror(fname);
                abort();
        }

        i = -1;
        while(!eof(fh)) {
                if (read(fh, (char *)&munou, (unsigned int)sizeof(MUNOU)) == -1)
 {
                        perror(fname);
                        abort();
                }
                strncpy(keyword[++i], munou.key_word, KEYLEN - 1);
                keyword[i][KEYLEN - 1] = '¥0';
        }
```

```
        maxrec = i;
        return(fh);
}


int             dic_close(int fd)
{
        if (close(fd) == -1) {
                perror(fname);
                abort();
        }
        return(TRUE);
}


void    rie_says(char *bun)
{
        printf("[%s] %s  %%¥n", mname, bun);
}


int             put_answer(int fn, int rec)
{
        if (lseek(fn, (long)(sizeof(MUNOU) * rec), 0) == -1) {
                perror(fname);
                puts("rie.dat / lseek");
                return(FALSE);
        }
        if (write(fn, (char *)&munou, (unsigned int)sizeof(MUNOU)) == -1) {
                perror(fname);
                puts("rie.dat / write");
                return(FALSE);

}


static char     *get_answer(int fn, int rec)
{
        static char *error = "error";

        if (lseek(fn, (long)(sizeof(MUNOU) * rec), 0) == -1) {
                perror(fname);
                return(error);
        }
        if (read(fn, (char *)&munou, (unsigned int)sizeof(MUNOU)) == -1) {
                perror(fname);
                return(error);
        }
        return(munou.answer);
}


int             find_word(int fn, char *dumkey)
{
        int             i, j, k;

        k = strlen(dumkey);
        strncat(dumkey, "              ", KEYLEN - k);
        for (i = KEYLEN - 1; i > KEYLEN - 8; i--) {
                for (j = 0; j <= maxrec; j++) {
                        if (strncmp(keyword[j], dumkey, i) == 0) {
                                rie_says(get_answer(fn, j));
                                return(j);
                        }
                }
        }
```

```
        return(FALSE);
}


static int      word_detect(char *bun)
{
        char    dwords[10][KEYLEN + 20], *dtkn;
        char    *delm = "　！？＃＄％＆’（）＝－，．、。・＿［］「」｜｛｝";
        int             i, k, l1, l2, kn;
        int             d_len[10];

        k=0;
        dtkn = jstrtok(bun, delm);
        while(dtkn != NULL) {
                strncpy(dwords[k], dtkn, KEYLEN + 19);
                dwords[k][KEYLEN + 19] = '¥0';
                d_len[k] = strlen(dwords[k]);
                k++;
                dtkn = jstrtok(NULL, delm);
        }
        kn = k;
        l1 = 0;
        l2 = 0;
        for(i = 0; i < k - 1; i++) {
                if(d_len[l1] < d_len[i+1]) {
                        l2 = l1;
                        l1 = i+1;
                }
        }
        strncpy(words, dwords[l1], KEYLEN + 17);
        strncat(words, "                                ", KEYLEN - strlen(dword
s[l1]) + 19);
        words[KEYLEN + 19] = '¥0';
        return(kn);
}



int             search_word(int fn, char *name)
{
        int     frec;
        char    bun[128];

        bun[0] = '¥0';
        while (1) {
                printf("¥r[%s] ", name);
                gets(bun);
                if (bun[0] != '¥0')
                        break;
                rie_says("からうちしないでね。");
                continue;
        }
        word_detect(bun);

        if ((frec = find_word(fn, words)) == FALSE) {
                bun[0] = '¥0';
                puts(" Not found !");
                printf("KeyWord : %s¥n", words);
                while (bun[0] == '¥0') {
                        printf("answer : ");
                        gets(bun);
                }
                printf(" OK ? (n/else) :");
                if (tolower(getch()) == 'n') {
                        puts(" Abort !");
                        return(frec);
                }
```

```
                strncpy(munou.key_word, words, KEYLEN - 1);
                strncpy(keyword[++maxrec],  words, KEYLEN - 1);
                strncpy(munou.answer, bun, ANSLEN - 1);
                munou.answer[ANSLEN - 1] = '¥0';
                put_answer(fn, maxrec);
                frec = maxrec;
                puts(" Completed !");
        }
        return(frec);
}

void    modify_rec(int fn, int frec)
{
        char    bun[128];

                puts(" Enter New Words and Answer !");

                bun[0] = '¥0';
                printf("key : ");
                gets(bun);
                word_detect(bun);
                if (bun[0] == '¥0')
                        strncpy(words, keyword[frec], KEYLEN - 1);
                printf("KeyWord : %s¥n", words);

                bun[0] = '¥0';
                while (bun[0] == '¥0') {
                        printf("answer : ");
                        gets(bun);
                }

                printf(" OK ? (n/else) :");
                if (tolower(getch()) == 'n') {
                        puts(" Abort !");
                        return;
                }

                strncpy(munou.key_word, words, KEYLEN - 1);
                strncpy(keyword[frec],  words, KEYLEN - 1);
                strncpy(munou.answer, bun, ANSLEN - 1);
                munou.answer[ANSLEN - 1] = '¥0';
                put_answer(fn, frec);
                puts(" Completed !");
                return;
}


void    delete_rec(int fn, int frec)
{
        char    *deleted = "9               ";

        strncpy(munou.key_word, deleted, KEYLEN - 1);
        strncpy(keyword[frec],  deleted, KEYLEN - 1);
        put_answer(fn, frec);
        puts(" Deleted !");
        return;
}


void    main(int argc, char **argv)
{
        int     fn, c, d, frec, dflag;
        char    name[40];

        if (argc == 1)
                        strcpy(name, "You");
```

```
	else
		strcpy(name, argv[1]);

	if ((mname = getenv("MUNOU")) == NULL)
		mname = "Rie";
	strcat(strcpy(fname, mname), ".dat");

	frec = 0;
	dflag =0;
	fn = dic_open();
	rie_says("ヘローゥ！　メンテしてねっ");

	putchar('>');
	while ((c = tolower(getch())) != 'q') {
		putchar(c);
		switch (c) {
			case 'k' :
				frec = 0;
				break;
			case 'j' :
				frec = maxrec;
				break;
			case 'h' :
				frec -= 1;
				if (frec < 0) {
					putchar(0x07);
					frec = 0;
					dflag = 1;
				}
				break;
			case 'l' :
				frec += 1;
				if (frec > maxrec) {
					putchar(0x07);
					frec = maxrec;
					dflag = 1;
				}
				break;
			case 'm' :
				modify_rec(fn, frec);
				break;
			case 'd' :
				printf(" Are you sure ? (y/else) : ");
				if ((d = tolower(getch())) == 'y') {
					putchar(d);
					delete_rec(fn, frec);
				} else
					puts(" Canceled !");
				break;
			case 's' :
				frec = search_word(fn, name);
				break;
			case '?' :
				puts("¥rHELP>  h: Lower Rec  l: Upper Rec  j: Bo
ttom Rec  k: Top Rec");
				puts("       s: Search KeyWord  m: Modify Rec  d
: Delete Rec  q: Quit");
				dflag = 1;
				break;
			default :
				putchar(0x07);
				dflag = 1;
				break;
		}
		if (dflag == 0 && maxrec != -1) {
			printf("¥r [%04d] key : %s ¥n",frec ,keyword[frec]);
			printf("   answer   : %s ¥n", get_answer(fn, frec));
```

```
		}
		putchar('¥r');
		putchar('>');
		dflag = 0;
	}
	putchar('¥n');
	rie_says("ばいばい。また、メンテしてね。");
	dic_close(fn);
```

プログラム４　★arg.c★

```
/********************************************

                args sample program

********************************************/

main(int argc, char **argv)
{
    int c, a_flag, b_flag, c_flag;
    char    *p;

    if (argc == 1) {
        usage();
        exit(2);
    }

    while (*(p = *++argv) == '-') {
        while ((c = *++p) != '¥0') {
                switch (tolower(c)) {
                case 'a':
                    a_flag = 1;
                    break;
                case 'b':
                    b_flag = 1;
                    break;
                case 'c':
                    c_flag = 1;
                    break;
                default:
                    cputs("Unknown option ¥n¥r");
                    usage();
                    exit(2);
                }
            }
    }
    exit(0);
}
```

3.3 モールス符号タッチコーダー

ここで紹介するtcode.cは3. 1で紹介したmorse.cの改造版です。morse.cでは、キーボードから入力されたコードをモールス・コードに置き換えて表示するものでした。tcode.cは、画面に表示するのではなくて、実際にPC-9801内蔵のベルを使って、音を出すというものです。このプログラムでは、コンソール入出力と、プロセス関係のシグナル関数を使用しています。

3.3.1 コンソール入出力関数の概要

コンソール入出力関数とは、MS-DOSの標準入出力関数のことで、conio.hで宣言されています。標準入出力に関しては、3. 1でストリーム入出力関数について説明してありますので、そちらを参照してください。

ストリーム入出力では、標準入出力以外のデバイスやファイルなどについても使用できましたが、コンソール入出力は標準入出力専用の関数です。したがって、ファイルのオープン・クローズの必要はありません。もちろん、コンソール入出力はリダイレクトできます。また、ポートの入出力関数も、このグループに入っています。

コンソール出力をするためには、次のような関数を使います。

cprintf　書式付き出力
putch　1bytes出力
cputs　1行出力

また、コンソールからの入力をするためには、次のような関数を使います。

cscanf　書式付き入力
getch　1bytes入力エコーバック無し
getche　1bytes入力エコーバックあり
cgets　1行入力

このように、ストリームのときと同じような関数が使えるようになっています。ただし、ストリーム入出力とコンソール入出力の関数は、実際には内部で行なっていることは違いますから、混在して同時に使わない方がよいでしょう。

outp関数とinp関数は、コンピュータのポートの入出力関数です。正いポート番号を指定しない場合には、とんでもない動作をすることがあるので、資料を揃えてから充分注意して使ってください。

3.3.2 シグナル関数の概要

MicrsoftCで書かれたプログラムは、特に処理を行なわない場合には、CTRL+Cでプログラムを終了します。しかし、途中で処理を中止したことによって、画面出力やファイルの入出力に悪影響を与えることがあります。たとえば、出力文字に特殊な色を付けていたりすると、終了しても文字に色が付いたままになってしまいます。

このような場合に使われるのがシグナル関数です。シグナル関数は、signal.hの中で宣言されています。signal関数では、CTRL+C以外にも異常終了が起こったときに、実行される関数を指定できます。その関数が後始末をすればよいわけです。また、異常終了を発生させるraise関数があります。

●コンソール入出力関数の使い方

tcode.cでは、stdio.hをインクルードしていません。ファイル入出力はコンソールのみですので、ストリームは使わずに、conio.hを使用しています。

プログラム自体はmorse.cを発展させたもので、getchar関数の代わりにgetch関数を、fputchar関数の代わりにputch関数を使用しています。

また、ベルを鳴らすために、outp関数を使用しています。PC-9801の場合には、ポートの37hに06hをoutするとベルが鳴ります。同様に、ポートの37hに06hをoutするとベルが止まります。あとは、ウェイトを変えてベルの長さを調節しているわけです。また、ベルの周波数を変えるために、ポートの3FBDhにも、outしています。

また、音を鳴らしているときにはストップキーがすぐに効きません。このため、関数bellの最後のループ内の先頭で、kbhit関数を呼んでいます。この関数は、キーボードが押されているかどうかのチェックをする関数ですが、これが入っていれば、ストップ

キーがすぐに効くようになります。

ストリーム入出力があるにも関わらずコンソール入出力を使うメリットは、何と言ってもでき上がったコードが小さいことです。プログラムを小さくまとめるためには、?printfなどの高機能な関数は使わないこと、そして、標準入出力関係であれば、コンソール入出力関数を使うことです。ただし、コンソール入出力関数では"¥n"だけでは改行するのみで、行頭に戻りません。この場合には、"¥n¥r"と指定しないといけません。

では、実際に大きさを比較してみましょう。

TCODE.EXEは、リスト通りのプログラムです。TCODES.EXEはストリーム入出力に書き変えたものです。TCODEP.EXEはストリーム入出力のうえに、puts関数ではなくprintf関数を使ったものです。(単位はbytes)

```
TCODE    EXE   10239
TCODES   EXE   11971
TCODEP   EXE   13991
```

このように、差は歴然としています。ただしprintfを使ってはいけないわけではありません。場合によります。必要なところを無理してprintfを使わずに、かえってわけのわからないプログラムを書くよりは、素直にprintfを使った方が良いのは当然です。

●シグナル関数の使い方

このプログラムでは、音が出ているときの文字をシアン色に変えています。したがって、途中ですストップキーが押されて、そのまま止まってしまうと文字の色がシアンのままになります。

そこで、signal関数を使ってストップキーが押されたときには、関数stopkeyに制御を移すようにしています。stopkey関数では、色を白に戻して異常終了しています。どのような割り込みシグナルかによって処理を変えることもできます。ここではSIGINT、すなわちCTRL+Cでの終了についてのみ記述しています。

また、atexit関数はプログラム終了時に実行される関数を指定できます。signal関数と違って正常終了時の関数ですが、ここでも色を戻すために使っています。atexit関数と同様の働きをするものに、onexit関数がありますが、ANSI案ではatexitとなっていますのでatexitの方がよいでしょう。

プログラム５　★tcode.c★

```
/*********************************************

              Morse Touch Corder

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include    <dos.h>
#include    <conio.h>
#include    <string.h>
#include    <signal.h>

#define MAX     256
#define TRUE    -1
#define FALSE   0
#define KEOF    0x1a

void main();
int bell(int, unsigned long);
void freq();
void speed(unsigned long *);
void bell_on(unsigned long);
void bell_off(unsigned long);
void key_buf_clr(void);
void usage(void);
void stopkey(void);

union REGS      inregs, outregs;

char    *mcode[MAX] = { "/", "/" ,".-..-." ,"/", "/", "/", ".=...", ".----.",
                        "-.--.", "-.--.-", "-..-", ".-.-.", "--..--", "-....-",
                        ".-.-.-", "-..-.", "-----", ".----", "..---", "...--",
                        "....-", ".....", "-....", "--...", "---..", "----.",
                        "---...", "---...", "/", "-....-", "/", "..--..", "/",
                        ".-", "-...", "-.-.", "-..", ".", "..-.", "--.",
                        "....", "..", ".---", "-.-", ".-..", "--", "-.", "---",
                        ".--.", "--.-", ".-.", "...", "-", "..-", "...-",
                        ".--", "-..-", "-.--", "--..", "/", "/", "/", "/", "/",
                        "/", ".-.-.-", "-.--.-", ".-..-.", "--..--", "---...",
                        ".---", "--.--", ".-", "..-", "-.---", ".-...", ".--",
                        "-..--", "--", ".--.", ".--.-", "--.--", ".-", "..-",
                        "-.---", ".-...", ".-..", "-.-..", "...-", "-.--",
                        "----", "-.-.-", "--.-.", "---.-", ".---.", "---.",
                        "-.", "..-.", ".--.", ".-.--", "..-..", ".-.", "-.-.",
                        "....", "--.-", "..--", "-...", "--..-", "--..", ".",
                        "-..", "-..-", "..-.-", "-", "..-.-", "-..-.", ".--",
                        "-..--", "--", "...", "--.", "-.--.", "---", ".-.-",
                        "-.-", ".-.-.", "..", "..--."
        };

char    *jcode = "-..---/";
char    *acode = "...-./";

int     w_flag = 0;
int     wabun = 0;


void stopkey(void)
{
    cputs("¥033[m");
    abort();
}
```

```
void main(int argc, char **argv)
{
    unsigned long   w;
    int             i, j, k, keyin[128], c;

    signal(SIGINT, stopkey);
    atexit(stopkey);

    w = (long)(0x0fff - (40 * 0x51));
    j = 0;
    cputs("CW Touch Coder by Naoshi Kimura¥n¥r");

    if (argc > 3) {
        usage();
        exit(2);
    } else if (argc > 1) {
        i = abs(atoi(argv[1]));
        if (0 < i && i < 50)
            w = (long)(0x0fff - (i * 0x51));
        else {
            usage();
            exit(2);
        }
    }

    cputs(" type ¥'ctrl + ¥¥¥' to change frequency ¥n¥r");
    cputs(" type ¥'ctrl + ^¥' to change speed ¥n¥r¥n¥r");

    while((c = getch()) != KEOF) {
        switch(c) {
        case 0x1c:
            freq();
            break;
        case 0x1e:
            speed(&w);
            break;
        case 0x08:
            if (j == 0)
                break;
            j--;
            cputs("¥008 ¥008");
            break;
        case 0x0d:
            putch(c);
            cputs("¥033[36m");
            for (k = 0; k < j; k++)
                bell(keyin[k], w);
            cputs("¥033[0m¥n¥r");
            j = 0;
            break;
        default:
            if (0x60 < c && c < 0x7b)
                c = toupper(c);
            if (c < 0x20 || (0x5f < c && c < 0xa1) || 0xdf < c)
                continue;
            putch(c);
            keyin[j++] = c;
            if (j > 79) {
                cputs("¥033[36m¥033[1A¥n¥r");
                for (k = 0; k < j; k++)
                    bell(keyin[k], w);
                cputs("¥033[0m¥n¥r");
                j = 0;
                continue;
            }
            break;
        }
```

```
    }
    exit(0);
}


int bell(int i, unsigned long w)
{
    char    tmp_code[20], dum_code[20];
    int     j, k;

    putch(i);
    if (0x5f < i) {
        wabun = 1;
        strcpy(tmp_code, mcode[i - 0x60]);
    } else {
        if (0x1f < i && i < 0x40 && wabun == 1)
            wabun = 1;
        else
            wabun = 0;
        strcpy(tmp_code, mcode[i - ' ']);
    }

    if (w_flag != wabun) {
        w_flag = wabun;
        strcpy(dum_code; tmp_code);
        if (w_flag == 0)
            strcpy(tmp_code, acode);
        else
            strcpy(tmp_code, jcode);
        strcat(tmp_code, dum_code);
    }

    for (j = 0; j < strlen(tmp_code); j++) {
        k = (int )tmp_code[j];
        kbhit();
        switch(k) {
        case '-':
             bell_on(w * 3);
             break;
         case '.':
             bell_on(w);
             break;
         case '=':
         default:
             bell_off(w * 2);
             break;
         }
         bell_off(w);
     }
     bell_off(w * 2);
     return(TRUE);
 }

 void freq()
 {
     unsigned int    sp, sph, spl;

     key_buf_clr();
     cputs("¥n¥rfreq 2457.5 / n [kHz] : ");
     scanf("%d", &sp);
     key_buf_clr();
     if (sp == 0)
         sp = 2500;
     spl = sp % 256;
     outp(0x3fdb, (char )spl);
     sph = sp / 256;
     outp(0x3fdb, (char )sph);
 }
```

```
void speed(unsigned long *w)
{
    unsigned int    dum = 0;

    key_buf_clr();
    while(dum < 1 || 50 < dum) {
        cputs("¥n¥rspeed (1 - 50) : ");
        scanf("%d", &dum);
        key_buf_clr();
    }

    *w = (long)(0x0fff - (dum * 0x51));
}


void bell_on(unsigned long k)
{
    int i, j;

    outp(0x37, 0x6);
    for (j = 0 ;j < 10; j++) {
        for (i = 0 ;i < k; i++)
            ;
    }

}


void bell_off(unsigned long k)
{
    int i, j;

    outp(0x37, 0x7);
    for (j = 0 ;j < 10; j++) {
        for (i = 0 ;i < k; i++)
            ;
    }
}


void key_buf_clr()
{
    inregs.h.ah = 0x0c;
    inregs.h.al = 0x00;

    intdos(&inregs, &outregs);
}


void usage()
{
    cputs("Unknown Optinon ¥n¥r");
    cputs("usage : tcode -n  [speed : 0 < n < 50]¥n¥r");
}
```

3.4 グラフィックローダー・セーバ

PC-9801にはグラフィックVRAMがあります。このグラフィックVRAMでは、4096色中の任意の16色を同時に表示できます(機種によっては不可能)。いろいろなメーカーからもグラフィックを扱うソフトが販売されています。ここでは、このグラフィックVRAMのイメージをファイルに書き込んだり、ファイルからVRAMに展開するプログラムを紹介します。ここではファイルはストリーム入出力関数を使い、データの移動にバッファ操作関数を使っています。

3.4.1 バッファ操作関数の概要

バッファ操作関数は、memory.hで宣言されています。これらの関数はメモリの内容をコピーしたり、比較したりするものです。

関数名はmem＊で始まっています。その後ろには、文字列操作関数str＊と似たような文字が続いており、同じような関数名になっています。たとえば、strncpy関数に対応するものとしてmemcpy関数があります。strcpyに対応ではありません。文字列操作関数は、ヌル文字'¥0'でターミネートされていたものを対象としていたのに対し、バッファ操作関数はヌル文字を意識しないものです。それ以外は大変に似た機能を持っています。したがって、文字列の長さの指定は必ず必要になるので、strncpyに対応しているわけです。

memcpy操作関数は、コピーを行うふたつの領域が重なっているとうまく動作しません。このような場合にはmemmove関数を使い処理してください。MicrsoftCバージョン4.0では重なっていたときにも大丈夫だったので、5.1では移植時には注意してください。

また、異なるセグメント間のデータのコピーをするものとしてmovedataがあります。

●バッファ操作関数の使い方

一般的なバッファ操作関数はstrn＊と似た動作をするので、ここでは取り上げません。

movedataは異なるセグメント間でもデータのコピーが可能なので、これを使った例を挙げました。

PC-9801ではグラフィックVRAMのRGBの各プレーンと16色表示のためのIプレーンを合成して1枚の絵として表示します。

青のプレーンはa8000Hから、赤のプレーンは、b0000Hから、緑のプレーンはb8000Hから、16色表示のためのIプレーンはe0000Hから始まっています。そのために、これらのデータは64Kbytesのセグメントの区切りを遙かに越えた巨大なデータとなります。したがって、このデータは一度データセグメントにコピーされてから書き込む必要が出てきます。この場面でmovedata関数が使われています。

gsave.cでは、これらの各プレーンのデータをそれぞれ、.B1、.R1、.G1、.E1という拡張子を付けてファイルに書き込みます。もちろん640×400ドット分のデータを扱うことができるようになっています。各プレーンは、セグメントアドレスと、ファイル名を引数としてとれば、同様な処理が可能です。したがってgsave関数では、これらを引数としてファイルをオープンしてmovedataの後、書き込みクローズするという処理を行なっています。mainでは引数を変えてgsave関数を呼び出すだけです。

gload.cは、gsave.cで書き込まれたファイルから、VRAMに絵を復元するプログラムです。基本的な構成はgsave.cと同じですが、16色パレットのイニシャライズをしています。ただし、PC-9801ではパレット情報がマシンから取得できません。デフォルトのパレット情報でない場合には、gsaveしたデータをgloadで読み込んでも、色が違うといった事態になります。これを解決したいときには、元絵を書く時点で、パレット情報を別ファイルに持っていないと不可能です。

プログラム6　★gsave.c★

```
/*********************************************

              Graphic save program

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <memory.h>
#include    <conio.h>
#include    <io.h>
#include    <dos.h>

#define TRUE    -1
#define FALSE    0
#define BUFSIZE 80

struct SREGS segregs;

int gsave(int , char *, char *);

void  main(int argc, char **argv)
{
    char        *t;

    if (argc < 2) {
        cputs("¥nProgramed by Naoshi Kimura¥n¥r");
        cputs("usage : gsave <filename> ¥n¥r");
        exit(2);
    }

    if ((t = strstr(argv[1], ".")) != NULL)
        *t = '¥0';

/*  BLUE  */
    if (gsave(0xa800, argv[1], ".B1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

/*  RED   */
    if (gsave(0xb000, argv[1], ".R1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

/*  GREEN  */
    if (gsave(0xb800, argv[1], ".G1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

/*  INTENS  */
    if (gsave(0xe000, argv[1], ".E1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

}
```

```
int gsave(int vseg, char *fn, char *ft)
{
    unsigned int    i;
    unsigned char   buf[BUFSIZE], str[80];
    FILE            *fp;

    if ((fp = fopen(strcat(strcpy(str, fn), ft), "wb")) == NULL)
        return(FALSE);
    segread(&segregs);

    for (i = 0;i < 0x7d00; i += BUFSIZE) {
        movedata(vseg, i, segregs.ds, (unsigned int)buf, 80);
        fwrite(buf, 1, BUFSIZE, fp);
    }
    if (fclose(fp) == EOF)
        return(FALSE);
    return(TRUE);
}
```

プログラム７　★gload.c★

```
/*******************************************

              Graphic load program

    Copyright (c) Naoshi Kimura   1988

*******************************************/

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <memory.h>
#include    <conio.h>
#include    <io.h>
#include    <dos.h>

#define TRUE    -1
#define FALSE    0
#define BUFSIZE 80

struct SREGS segregs;

void         g_init();
int          gload(int, char *, char *);
void         apal(int , int);

#pragma intrinsic(outp)

void apal(int pal, int col)
{
    outp(0xa8, pal);
    outp(0xaa, (col >> 8) & 0xf);
    outp(0xac, (col >> 4) & 0xf);
    outp(0xae, col & 0x0f);
}

void apalinit(void)
{
    static int ipal[16] = {
        0x000, 0x00f, 0x0f0, 0x0ff, 0xf00, 0xf0f, 0xff0, 0xfff,
        0x000, 0x008, 0x080, 0x088, 0x800, 0x808, 0x880, 0x888,
    };

    int i;

    for (i = 0; i < 16; i++)
        apal(i, ipal[i]);
}


void g_init()
{
    union REGS  reg;

    outp(0xa4, 0);
    outp(0xa6, 0);
    outp(0x6a, 0x01);

    apalinit();

    reg.h.ah = 0x42;
    reg.h.ch = 0xc0;
    int86(0x18, &reg, &reg);

    reg.h.ah = 0x40;
```

```
    int86(0x18, &reg, &reg);

    outp(0x6a, 0x01);
}


void main(int argc, char **argv)
{
    char        *t;

    if (argc < 2) {
        cputs("¥nProgramed by Naoshi Kimura¥n¥r");
        cputs("usage : gload <filename> ¥n¥r");
        exit(2);
    }
    g_init();

    if ((t = strstr(argv[1], ".")) != NULL)
        *t = '¥0';

/*  BLUE  */
    if (gload(0xa800, argv[1], ".B1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

/*  RED  */
    if (gload(0xb000, argv[1], ".R1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

/*  GREEN  */
    if (gload(0xb800, argv[1], ".G1") != TRUE) {
        perror(argv[1]);
        exit(2);
    }

/*  INTENS  */
    if (gload(0xe000, argv[1], ".E1") != TRUE) {
        perror(argv[1]);
        outp(0x6a, 0x00);
        cprintf("can not found %s.E1 !¥n", argv[1]);
        exit(2);
    }

}
int gload(int vseg, char *fn, char *ft)
{
    unsigned int    i, j;
    unsigned char   buf[BUFSIZE], str[80];
    FILE            *fp;


    if ((fp = fopen(strcat(strcpy(str, fn), ft), "rb")) == NULL)
        return(FALSE);
    segread(&segregs);

    for (i = 0; i < 0x7d00; i += BUFSIZE) {
        fread(buf, 1, BUFSIZE, fp);
        movedata(segregs.ds, (unsigned int)buf, (unsigned int)vseg, i, 80);
    }
    if (fclose(fp) == EOF)
        return(FALSE);
    return(TRUE);
}
```

3.5 日本語時間表示プログラム

現在の時刻を、日本語表記で表示するプログラムです。MicrsoftCではパソコンのシステムクロックから日付や時刻を得る方法がいくつかあります。一般にはtime.hで宣言されている関数を使いますが、MS-DOSのファンクションコールを直接使う_dos関数でも、この機能を実現できます。そこで、この別々のふたつの方法によって、同じ機能を持ったプログラムを作成してみました。また、なるべくプログラムを小さくする方法についても考えてみました。

3.5.1 時間関数の概要

時間関数は、time.h、timeb.h、utime.hで宣言されており、パソコンのシステムクロックから現在の時刻を得て、その内容を変換したり格納するものです。

時刻を取得する関数はtime関数です。この関数は1970年1月1日からの経過時間を秒で返します。gmtime、localtime関数は秒単位で得られた時間を、年月日・時分秒形式の構造体tmに変換します。構造体tmはtime.hで、次のように定義されています。

```
struct tm {
    int tm_sec;       秒
    int tm_min;       分
    int tm_hour;      時
    int tm_mday;      日
    int tm_mon;       月
    int tm_year;      年
    int tm_wday;      曜日
    int tm_yday;      年の通算日
    int tm_isdst;     夏期時間
};
```

asctime、ctime関数は時刻情報を文字列に変換します。asctime関数は構造体tmの時刻データを文字列に変換します。ctime関数はtime関数により得られる秒単位情報を文字列に変換します。

mktime関数は、構造体tmの情報をtime関数で得られる秒数値に変換します。

time、gmtime、localtime、asctime、ctime、mktime関数はtime.hで宣言されています。

ftime関数は現在のシステム時刻を構造体timebに返します。構造体timebはsys/timeb.hで次のように定義されています。

```
struct timeb {
    time_t time;
    unsigned short millitm;
    short timezone;
    short dstflag;
};
```

utime関数はファイルの更新日時を変更する関数です。utime関数で使用する構造体utimebufはsys/utime.hで次のように定義されています。

```
struct utimbuf {
    time_t actime;        最終参照時刻
    time_t modtime;       最終更新時刻
};
```

modtimeは最終更新時刻を示す1970年1月1日からの経過秒数です。actimeは最終参照時刻を示すものですがMS-DOSではサポートされていませんので、設定は無視されます。

●時間関数の使い方

このプログラムは、ごく単純な手法を用いて時刻の日本語表示を行っています。time関数により得られた秒単位の時刻情報をlocaltime関数でtm構造体に変換します。その後tm構造体の各要素を日本語文字列に変換し、printf関数で表示します。数値を日本語文字列へ変換する関数jnumberは、sprintf関数により数字→アスキー文字列変換を行い、その後hantozen関数を用いて、アスキー文字列→日本語文字列変換を行っています。

●シェイプアップ

jtime1.cの書き方はごく普通の手法によるもので、ある意味では良いプログラムと言えるでしょう。

jtime1.cではprintf関数やsprintf関数など高機能

なライブラリ関数を用いています。これらの関数は浮動小数点フォーマットの表示など、かなり高機能で、これらの関数を使用するとサイズも相当に大きくなってしまいます。jtime1.cでは文字列を連続的に表示させるためにprintfを、数値を文字列に変換するためにsprintfを使用しています。この程度の処理ならば、自分でプログラムを書いてしまった方が小さくなります。jtime2.cでは、printf、sprintf関数を使用しないで同じ機能を実現したものです。jtime3.cではtime関数すらも使わないで実現したものです。hantozen関数も使用していません。実際にコンパイル・リンクしてみるとサイズの差が明確にわかるでしょう。(単位はbytes)

JTIME1.EXE	10387
JTIME2.EXE	5645
JTIME3.EXE	2917

jtime3.cは、まだ小型化の余地があります。どこまで小さくなるかチャレンジしてみるのも面白いと思います。

プログラム８　★jtime1.c★

```
/**********************************************

                日本語 time

                   with
 printf/sprintf and standard time functions

    Copyright (c) Naoshi Kimura   1988

**********************************************/

#include <stdio.h>
#include <jstring.h>
#include <time.h>

char *jwdaytab[7] = { "日", "月", "火", "水", "木", "金", "土" };

void jnumber(unsigned char *jbuf, int n)
{
    char numbuf[16];
    char *p;
    unsigned short sj;

    sprintf(numbuf, "%d", n);
    for (p = numbuf; *p != '¥0'; p++) {
        sj = hantozen(*p);
        *jbuf++ = (unsigned char)(sj >> 8);
        *jbuf++ = (unsigned char)(sj & 0xff);
    }
    *jbuf = '¥0';
}

int main(void)
{
    time_t t;
    struct tm *tm;
    unsigned char jyear[16];
    unsigned char jmonth[8];
    unsigned char jmday[8];
    unsigned char jhour[8];
    unsigned char jmin[8];
    unsigned char jsec[8];

    time(&t);
    tm = localtime(&t);
    jnumber(jyear, tm->tm_year + 1900);
    jnumber(jmonth, tm->tm_mon + 1);
    jnumber(jmday, tm->tm_mday);
    jnumber(jhour, tm->tm_hour % 12);
    jnumber(jmin, tm->tm_min);
    jnumber(jsec, tm->tm_sec);

    printf("西暦%s年　%s月%s日%s曜日　午%s%s時%s分%s秒¥n",
        jyear, jmonth, jmday, jwdaytab[tm->tm_wday],
        tm->tm_hour >= 12 ? "後" : "前",
        jhour, jmin, jsec);
    return 0;
}
```

プログラム９　★jtime2.c★

```
/*******************************************

                日本語 time

    with cputs and standard time functions

    Copyright (c) Naoshi Kimura   1988

*******************************************/

#include <conio.h>
#include <string.h>
#include <jstring.h>
#include <time.h>

char *jwdaytab[7] = { "日", "月", "火", "水", "木", "金", "土" };

void jnumber(unsigned char *jbuf, int n)
{
    char numbuf[16];
    char *p;
    unsigned short sj;

    p = numbuf;
    numbuf[0] = '0';
    numbuf[1] = '¥0';
    while (n > 0) {
        *p++ = '0' + (n % 10);
        *p = '¥0';
        n /= 10;
    }
    strrev(numbuf);
    for (p = numbuf; *p != '¥0'; p++) {
        sj = hantozen(*p);
        *jbuf++ = (unsigned char)(sj >> 8);
        *jbuf++ = (unsigned char)(sj & 0xff);
    }
    *jbuf = '¥0';
}

int main(void)
{
    time_t t;
    struct tm *tm;
    unsigned char jyear[16];
    unsigned char jmonth[8];
    unsigned char jmday[8];
    unsigned char jhour[8];
    unsigned char jmin[8];
    unsigned char jsec[8];

    time(&t);
    tm = localtime(&t);
    jnumber(jyear, tm->tm_year + 1900);
    jnumber(jmonth, tm->tm_mon + 1);
    jnumber(jmday, tm->tm_mday);
    jnumber(jhour, tm->tm_hour % 12);
    jnumber(jmin, tm->tm_min);
    jnumber(jsec, tm->tm_sec);

    cputs("西暦");
    cputs(jyear);
    cputs("年　");
    cputs(jmonth);
    cputs("月");
    cputs(jmday);
    cputs("日");
    cputs(jwdaytab[tm->tm_wday]);
    cputs("曜日　午");
    cputs(tm->tm_hour > 12 ? "後" : "前");
    cputs(jhour);
    cputs("時");
    cputs(jmin);
    cputs("分");
    cputs(jsec);
    cputs("秒¥r¥n");
    return 0;
}
```

プログラム10　★jtime3.c★

```
/*********************************************

                 日本語 time

   with cputs and _dos_time/_date functions

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include <conio.h>
#include <dos.h>

char *jwdaytab[7] = { "日", "月", "火", "水", "木", "金", "土" };

void jnumber(unsigned char *jbuf, int n)
{
    char numbuf[16];
    char *p;

    p = numbuf;
    if (n > 0)
        while (n > 0) {
            *p++ = n % 10;
            n /= 10;
        }
    else
        *p++ = 0;
    while (--p >= numbuf) {
        *jbuf++ = 0x82;
        *jbuf++ = 0x4f + *p;
    }
    *jbuf = '¥0';
}

int main(void)
{
    struct dostime_t dt;
    struct dosdate_t dd;
    unsigned char jyear[16];
    unsigned char jmonth[8];
    unsigned char jmday[8];
    unsigned char jhour[8];
    unsigned char jmin[8];
    unsigned char jsec[8];

    _dos_getdate(&dd);
    _dos_gettime(&dt);
    jnumber(jyear, dd.year);
    jnumber(jmonth, dd.month);
    jnumber(jmday, dd.day);
    jnumber(jhour, dt.hour % 12);
    jnumber(jmin, dt.minute);
    jnumber(jsec, dt.second);
    cputs("西暦");
    cputs(jyear);
    cputs("年  ");
    cputs(jmonth);
    cputs("月");
    cputs(jmday);
    cputs("日");
    cputs(jwdaytab[dd.dayofweek]);
    cputs("曜日  午");
    cputs(dt.hour > 12 ? "後" : "前");
    cputs(jhour);
    cputs("時");
    cputs(jmin);
    cputs("分");
    cputs(jsec);
    cputs("秒¥r¥n");
    return 0;
}
```

3.6 迷路作成プログラム

PC-9801シリーズは640×400ドットのカラーグラフィック機能を持っています。従来のMicrsoftCではグラフィック機能を使用するライブラリ関数は付属していませんでした。QuickCやMicrsoftCバージョン5.1からはPC-9801シリーズ用のグラフィックライブラリが付属するようになりました。ここでは、サンプルプログラムmaze.cを使用してMicrsoftCグラフィック関数の使い方を説明します。

3.6.1 mazeの起動方法

このプログラムmazeは、グラフィック画面いっぱいに大きな迷路を描画します。

A>maze

と入力すると、画面に迷路を作成し始め、迷路が完成するとビープ音を鳴らします。何かキーを押すとグラフィックを非表示にして、MS-DOSに戻ります。

3.6.2 グラフィック関数の概要

グラフィック関数では、論理座標、パレット、クリッピング領域などの機能を備えています。これらの概念はN88BASICなどにもあるものですが、MicrsoftCグラフィックライブラリでは機能の実現方法が独特のものとなっています。これはIBM-PCのグラフィックライブラリとの互換性を求めたためで、PCシリーズでグラフィックを使用するプログラムをPC-9801シリーズに移植したり、その逆の移植を行う際に、なるべく負担を軽くしようとする配慮です。

3.6.2.1 スクリーンモード

使用できる画面モードは、機種やオプションボードの有無によって変わってきます。PC-9801シリーズの場合、アナログディスプレイであるどうか、さらに16色表示ボードがあるかどうかにより、使用できるモードが変わります。御自分のPC-9801シリーズがどのようなグラフィックモードを使用できるかは、本体のマニュアルを参照してください。

PC-9801シリーズで使用できるビデオモードは以下のとおりです。

_DEFAULTMODE
_98TEXT80
80桁×25行の日本語テキスト表示を行うモードです。このモードではグラフィック表示を行いません。

_98RESSCOLOR
デジタルRGBの8色とテキストのスーパーインポーズを行うモードで、グラフィックは640×400ドット使用可能です。

_98RESS8COLOR
アナログRGBの4096色中8色とテキストのスーパーインポーズを行うモードで、グラフィックは640×400ドット使用できます。アナログRGBモニタが必要です。

_98RESS16COLOR
アナログRGBの4096色中16色とテキストのスーパーインポーズを行うモードで、グラフィックは640×400ドット使用できます。アナログRGBモニタと16色ボードが必要です。

_98RESCOLOR
デジタルRGBの8色のグラフィック表示を行うモードで、640×400ドット使用できます。テキスト画面は表示されません。

_98RES8COLOR
アナログRGBの4096色中8色のグラフィック表示を行うモードで、640×400ドット使用できます。テキスト画面は表示されません。アナログRGBモニタが必要です。

_98RES16COLOR
アナログRGBの4096色中16色のグラフィック表示を行うモードで、640×400ドット使用できます。テキスト画面は表示されません。アナログRGBモニタと16色ボードが必要です。

これらの画面モードを設定するには、_setvideomode関数を使用します。以上の説明からわかるように「白黒モード」はサポートされていません。デジタル8色モードなどを使用すれば、パレット操作によって白黒表示を行うこともできます。また、640×200ドットの解像度もサポートされていません。

実行機種がどのようなグラフィックモードをサポートできるかは、_getvideoconfig関数で知ることができます。

3.6.2.2 座標系

MicrsoftCのグラフィックライブラリでは「物理座標系」と「論理座標系」のふたつの座標系の概念を採り入れています。

「物理座標系」はシステムが持つグラフィック画面全体を指すものです。画面の左上を(0, 0)とした絶対座標です。PC-9801シリーズの場合、画面右下の物理座標は(639, 399)となります。この原点は変更できません。言い換えるならば、どのような場合にも、画面左上の物理座標は(0, 0)であることが保証されていることになります。

一方、「論理座標系」の原点は画面上の任意の位置に設定することができます。MicrsoftCのグラフィックライブラリの描画命令はすべて論理座標系によって開始位置、終了位置等を指定するようになっています。初期状態の論理座標の原点は物理座標の原点と一致しています。つまり、論理座標系の設定を行わない限り、論理座標と物理座標を意識する必要はありません。

論理座標系の設定を行うには、_setlogorg関数を使用します。たとえば、

```
_setlogorg(100, 100);
_setpixel(0, 0);
```

を行うと、論理座標原点を物理座標(100, 100)に置き、論理座標原点のドットをセットします。つまり、物理座標(100, 100)に論理座標原点(0,0)がセットされたことになります。

論理座標・物理座標間の座標変換関数として、_getlogcoord、_getphyscoord関数が用意されています。

3.6.2.3 色コード、パレット、ラインスタイル

MicrsoftCグラフィックライブラリの描画命令は、カレントカラーで描画を行います。カレントカラーの初期値はPC-9801シリーズの場合7(白)です。

カレントカラーを変更するには_setcolor関数を使用します。

```
_setcolor(0);
_setpixel(0, 1);
```

この例では、論理座標の(0, 1)に黒の点を描画します。このように、描画を行う前にあらかじめ色を設定しておきます。

現在設定されている色は_getcolorにより知ることができます。

画面クリアはバックグラウンドカラーで領域を塗りつぶします。バックグラウンドカラーはカレントカラーと同様に、_setbkcolor関数、_getbkcolor関数で設定・取得できます。

ラインスタイルが設定されていると、設定されたラインスタイルによって描画が行われます。ラインスタイルを設定・取得する関数は_setlinestyle、_getlinestyle関数です。初期状態ではラインスタイルは設定されていません。描画は直線で行われます。

パレットを変更するには_remappalette関数、_remapallpalette関数を使用します。たとえば、

```
_remappalette(1, 7);
```

は、青で表示されていた(色コード1で表示されていた)部分をすべて白(パレットコード7に)変更します。このように_remappalette関数はひとつのパレットを変更しますが、_remapallpalette関数は引数を配列で与え、すべてのパレットを一括して変更します。

MicrsoftCグラフィックライブラリは、グラフィックのみでなくテキスト表示を行う関数もサポートしています。上述の色指定関数はグラフィック表示に関するもので、テキスト表示には影響を及ぼしません。テキスト表示の色設定・取得には_settextcolor、_gettextcolor関数を用います。ここで行う色設定

は_outtext関数等、グラフィックライブラリのテキスト表示関数に対して有効です。コンソール出力時の表示色を変更するものではありません。

3.6.2.4 描画関数

MicrsoftCグラフィックライブラリでは、

_clearscreen	画面消去
_setpixel	点
_lineto	直線
_rectangle	長方形
_ellipse	楕円(含む円)
_arc	楕円弧(含む円弧)
_pie	扇型
_floodfill	塗りつぶし
_tilepaint	中間色塗りつぶし

の描画関数を持っており、パラメータを指定するだけで簡単に描画が可能です。直線の描画はカレントポジションから目的の論理座標までの描画を行います。始点となるカレントポジションは_moveto関数で指定されます。

```
_moveto(0, 0);
_lineto(639, 399);
```

この例では画面一杯に斜めの線を描画します。描画終了後のカレントポジションは(639, 399)になります。

楕円や、楕円弧、扇型の描画では、外接する長方形の対角の論理座標を与えます。中心点や半径、角度などを与えるのではありませんので、注意が必要です。

```
_ellipse(_GFILLINTERIOR, 0, 0, 100, 100);
```

これは、画面左上に円を描画します。個々の詳しい説明はライブラリリファレンスを参照してください。

3.6.2.5 イメージ転送関数

画面上の特定の領域をデータとして転送する関数も用意されています。

_getimage	イメージ取得
_putimage	イメージ描画
_imagesize	イメージサイズの取得

_getimage関数は、取得したい領域の対角の論理座標を与え、バッファに転送します。転送するバッファにはグラフィックデータが格納できるだけの充分な大きさがなければなりません。バッファに必要な大きさは_imagesize関数によって知ることができます。転送前に_imagesize関数で得られたバイト数分のバッファをアロケートして、そこに転送する方法が現実的でしょう。

_putimage関数は_getimage関数で得られたグラフィックデータを画面上に表示するものです。表示する領域の左上の論理座標を与えることにより、描画が行われます。

●グラフィック関数

グラフィック関数の概要はこれまで述べたとおりです。ただし、これですべての関数を紹介したわけではないので、詳細は関数リファレンスを参照してください。

それではサンプルプログラムmaze.cの中でグラフィック関数がどのように用いられているかを説明しましょう。

関数mainでは乱数の初期化、グラフィック画面の初期化・迷路作成ルーチンの呼び出しを行っています。

```
_setvideomode(_98RESSCOLOR);
```

で、画面モードをデジタルRGB8色モードに設定し、

```
_clearscreen(_GCLEARSCREEN);
```

により、画面消去を行っています。カーソル表示は必要無いので、

```
_displaycursor(0);
```

で、カーソルを消去しています。迷路の描画は青プレーン上です。しかし、実際に青が表示されたならば見にくいと思われるため、_remappalette関数を用いて、書き込むグラフィック表示を色コード1で白で表示するように変更しています。

```
_remappalette(1, _98WHITE);
```

以上の初期化が終了すると、関数mazeを呼び出し、実際の迷路の描画を開始します。迷路の描画が終了すると、ベルを鳴らし1文字入力を待ちます。

```
putch(7);
getch();
```

その後、カーソル状態、パレット状態、ビデオモードを元に戻しプログラムを終了します。

```
_displaycursor(1);
_remappalette(1, 1L);
_setvideomode(_DEFAULTMODE);
cputs("¥033 [2J");
```

最後に行っているcputs("¥033 [2J"); ではエスケープシーケンスを用いて画面消去を行います。cputs() はコンソール出力関数です。これはMicrsoftCグラフィックライブラリが25行目に表示されるステータスラインを消去したままにしてしまうため、再表示を行わせるものです(もともと表示されていない場合は消えたままになります)。

以上述べたグラフィック関数は、モード設定に関するもので実際の描画は行いません(_clearscreenは除く)。それでは、グラフィック描画を行っている関数を見てみましょう。

関数drawboxは、迷路の外枠を描画する関数です。

```
_setcolor(1);
```

で、描画色を青に設定します。(実際にはパレットにより白で表示されます)

```
_rectangle(_GBORDER, 0, 0, XMAXWALL,
YMAXWALL);
```

画面一杯に四角形を描画します。XMAX-WAWLLは639、YMAXWALLは399に#defineされています。これは迷路の外枠となります。

```
_setcolor(0);
```

このままでは、迷路の入口と出口がありませんから、左上と右下のあたりに出入口を作ります。外枠に穴をあける形となるため、描画色を黒に設定し、

```
_setpixel(0, 1);
_setpixel(XMAXWALL, YMAXWALL-1);
```

で、ドットをセット、出入口を付けました。

迷路作成のアルゴリズムは、最初に外枠を作り、ランダムに壁を伸ばして行く手法です。伸ばした壁は他の壁と接触しないようにします。そうすればどこか1本だけ通れる道ができるはずです。他の多くの関数は_linetoを用いたランダムな壁の描画を行うものです。これはグラフィック関数の処理と直接関係無いので割愛します。

プログラム11　★maze.c★

```
/*********************************************

              Maze Make Program

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <conio.h>
#include <time.h>

#define XMAXWALL     638
#define YMAXWALL     398
#define XVWALLS      (XMAXWALL / 2 - 1)
#define YHWALLS      (YMAXWALL / 2 - 1)
#define RWALLS       (XVWALLS * 2 + YHWALLS * 2)
#define GROW_COUNT   RWALLS
#define FILL_COUNT   60000

struct xy {
    int x;
    int y;
};

int random(int n)
{
    int r;

    r = rand() % n;
    return r;
}

int randomwalk(int *x, int *y)
{
    static struct xy xymap[4] = {
        -2, 0, 2, 0, 0, -2, 0, 2
    };
    struct xy *xyp;
    int xx, yy;
    int i;

    i = 0;
    while (1) {
        xyp = xymap + random(4);
        xx = *x + xyp->x;
        yy = *y + xyp->y;
        if (i++ > 6)
            return -1;
        if (xx < 0 || xx > XMAXWALL)
            continue;
        if (yy < 0 || yy > YMAXWALL)
            continue;
        if (_getpixel(xx, yy) != 0)
            continue;
        break;
    }
    _lineto(xx, yy);
    *x = xx;
    *y = yy;
    return 0;
}
```

```
void rwalltoxy(int *x, int *y, int r)
{
    if (r < XVWALLS) {
        *x = r * 2 + 2;
        *y = 0;
        return;
    }
    r -= XVWALLS;
    if (r < XVWALLS) {
        *x = r * 2 + 2;
        *y = YMAXWALL;
        return;
    }
    r -= XVWALLS;
    if (r < YHWALLS) {
        *x = 0;
        *y = r * 2 + 2;
        return;
    }
    r -= YHWALLS;
    *x = XMAXWALL;
    *y = r * 2 + 2;
    return;
}


void grawwall(void)
{
    int i, x, y;

    _setcolor(1);
    for (i = 0; i < GROW_COUNT; i++) {
        rwalltoxy(&x, &y, random(RWALLS));
        _moveto(x, y);
        while (randomwalk(&x, &y) == 0);
    }
}


void fillwall(void)
{
    int x, y;
    unsigned u;

    for (u = 0; u < FILL_COUNT; u++) {
        x = random(XVWALLS) * 2 + 2;
        y = random(YHWALLS) * 2 + 2;
        if (_getpixel(x, y) == 0)
            continue;
        _moveto(x, y);
        while (randomwalk(&x, &y) == 0);
    }

}


void finishmaze(void)
{
    int i, j, x, y;

    for (j = 0; j < YHWALLS; j++)
        for (i = 0; i < XVWALLS; i++) {
            x = i * 2 + 2;
            y = j * 2 + 2;
```

```
            if (_getpixel(x, y) != 0)
                continue;
            _moveto(x, y);
            if (random(2))
                _lineto(x - 2, y);
            else
                _lineto(x, y - 2);
        }
}


void drawbox(void)
{
    _setcolor(1);
    _moveto(0, 0);
    _lineto(0, YMAXWALL);
    _lineto(XMAXWALL, YMAXWALL);
    _lineto(XMAXWALL, 0);
    _lineto(0, 0);
    _setcolor(0);
    _setpixel(0, 1);
    _setpixel(XMAXWALL,  YMAXWALL - 1);
}

void maze(void)
{
    drawbox();
    grawwall();
    fillwall();
    finishmaze();
}


void main(void)
{
    time_t t;

    time(&t);
    srand((int)(t & 0xffff));
    _setvideomode(_98RESSCOLOR);
    _clearscreen(_GCLEARSCREEN);
    _displaycursor(0);
    _remappalette(1, _98WHITE);
    maze();
    putch('¥a');
    getch();
    _displaycursor(1);
    _setvideomode(_DEFAULTMODE);
    cputs("¥033[2J");
}
```

3.7 ターミナルプログラム

ここでサンプルとして取り上げるターミナルプログラムは、パソコン通信やホストコンピュータと接続するために用いる無手順ターミナルプログラムです。ターミナルプログラムはひとつ手に入れておけば、PDS(パブリックドメインソフト:パソコン通信などを用いて無料配布されるソフト)の良質のプログラムを手に入れることができます。このサンプルでは、きわめて基本的な機能を持つターミナルプログラムの書き方を通して、ハードウェア割り込みの処理方法、周辺機器のドライブ方法、MS-DOS特有の機能・関数の用例を挙げて行きます。

3.7.1 terminalの起動方法

このターミナルプログラムは、

terminal [-b{75 | 150 | 300 | 600 | 1200 | 2400 | 4800 | 9600}] [-c {7 | 8}] [-p {n | o | e}] -s {1 | 1.5 | 2}

で実行されます。-bの後の数値はボーレイト、-cの後の数値はキャラクタビット数、-pの後の数値はパリティビット、-sの後の数値はストップビット長です。通信内容を記録したい場合はファイルにリダイレクトしてください。

例: terminal -b2400 -c8 -pn -s1 > logfile.txt

3.7.2 MS-DOS関数の概要

大部分のC言語ライブラリは、UNIXとの互換性を求めて作られています。しかし、より細かい処理をさせたい場合にはMS-DOS依存の関数が役に立ちます。

たとえば、1文字入力で^Cコードを受け取らなければならない場合などが挙げられるでしょう。3.3.2 シグナル関数の概要で説明されているように、^Cコードをシグナル処理として実現する方法もあります。しかし、MS-DOSのファンクションコールを用いて入力をRAWモードに設定すれば、デバイスリードのファンクションが^Cによる中断を行わなくなります。

このように微妙な入出力の設定は標準的なC言語ライブラリには付属していません。そこで、MS-DOSのファンクションコールを直接呼び出すことになります。

MS-DOSファンクションコールを直接呼び出す関数には、intdos/intdosx関数があります。これらの関数を用いると、あたかも機械語でCPUのレジスタ類を設定し、ファンクションコールを行うかのような細かい処理ができるようになります(リファレンスintdos/intdosx参照)。

その他、_dosで始まる関数名のライブラリ関数では、MS-DOSファンクションコールレベルでの操作が可能になっています。intdos/intdosx関数を用いるよりも読みやすく、簡単にMS-DOSの機能を用いることができるます。ただし、すべてのMS-DOSファンクションコール呼び出しが_dosでサポートされているわけではありませんので、サポートされない部分についてはintdos/intdosx関数を使用することになります。

3.7.3 割り込み処理

MicrsoftCでは割り込み処理ルーチンをC言語で書けるようにinterrupt型の関数が用意されています。interrupt型の関数は、関数の入口ですべてのCPUレジスタの値を退避し、割り込みルーチンとしてC言語を使用できるように環境設定を行います。

interrupt型の関数は戻り値を持つことができません。つまり、すべてvoid型となります。また、割り込み処理ルーチンはセグメント:オフセットを割込ベクタテーブルに登録して呼び出しを行いますので、特殊なfar呼び出しと考えられます。ですから、interrupt型関数はfar型関数でなければなりません。また、interrupt型関数は引数を取ることができません。以上のことからinterrupt型関数の宣言は次のようになるのが一般的です。

```
void interrupt far 関数名(void);
```

●MS-DOS関数(デバイス入出力)の使い方

dosfunc.cではMS-DOS関数が多用されています。

ioctl_raw関数ではhandleで指定されたファイルハンドルのRAWモード・COOCKEDモードを切り換えるものです。この機能を実現するためには、MS-DOSファンクションコールの0x4400，0x4401を用いなければなりません。IOCTLデータの取得を行い、RAWモードのフラグをセット・リセットし、IOCTLデータの書き込みを行っています。IOCTLに関するMS-DOSファンクションコールについては、MS-DOSプログラマーズリファレンスを参照してください。

ioctl_isready関数はhandleで指定されたデバイスが読みだし可能であるかを調査します。そのためMS-DOSファンクションコール0x4406を使用しています。これはキーボードが押されたかどうかのチェックに使用しています。キーボードが押されたかどうかをチェックする関数にkbhit()がありますが、kbhit()関数は必ず^Cのチェックを行いますので、今回は使用できません。

readch関数はhandleで指定されたデバイスから1文字入力を行います。この際_dos_read関数を用いてMS-DOSファンクションコールを直接呼び出しています。

writech関数はhandleで指定されたデバイスに1文字出力を行います。この際_dos_write関数を用いてMS-DOSファンクションコールを直接呼び出しています。

getbreakflag関数はブレークフラグ(COMMAND.COM内部命令BREAKで参照・変更できる)の取得を行います。setbreakflag関数ではブレークフラグの設定を行います。ブレークフラグがonであるとMS-DOSファンクションコールの入口で必ず^Cチェックを行うため、ターミナルプログラムではOFFの状態にしておく必要があります。取得にはintdos関数を使用して、MS-DOSファンクションコール0x33を呼び出しています。

●割り込み処理

rsio.cでは割り込み処理を行っています。

rsintr関数はRS-232C回線から1文字受信があるごとに呼び出されます。この関数ではRS-232Cインターフェイスからinp()関数で1文字読み取りを行い、バッファに追加する動作を実行しています。ハードウェア割り込みルーチンの終了を示すEOIを割り込みコントローラに送出して、割り込みルーチンを終了しています。バッファはFIFO形式(先入れ先出し)形式で管理されています。バッファがオーバーフローすると古いデータから捨てられて行くことになります。

ハードウェア割り込みによりバッファに格納されたデータは、rs_getch関数で読み出すことができます。バッファから1文字取得するだけの関数です。

rs_putch関数は、文字をRS-232C回線に出力します。RS-232Cに文字が出力できるようになるまで待った後、outp関数で文字出力を行います。

sio_portinit関数はRS-232CインターフェイスLSIの初期化、speed_portinit関数はボーレイト設定のためのインターバルタイマの初期化、cpuclock関数はCPUのクロックスピードの取得を行う関数です。これらの関数はrs_init関数で内部的に使用されています。

rs_init関数はRS-232Cの初期化を行い、ハードウェア割り込み処理ルーチンをrsintr関数に設定し、割り込み許可を行います。ここで用いられている_dos_getvect/_dos_setvect関数は割り込みベクタの取得・設定のためのライブラリ関数です。

rs_release関数はRS-232C割り込みを禁止し、割り込みベクタを呼び出し前の状態に戻します。rs_init関数を用いたプロセスでは、終了前に必ずこの関数を呼んでください。呼ばずに終了した場合、暴走する危険があります。

プログラム12　★terminal.h★

```
/*********************************************

              terminal.h

*********************************************/

extern  int ioctl_raw(int, int);
extern  int ioctl_isready(int);
extern  int readch(int);
extern  void writech(int ,int);
extern  int getbreakflag(void);
extern  void setbreakflag(int);

#define STDIN   0
#define STDOUT  1
#define STDERR  2
```

プログラム13　★rs232c.h★

```
/*********************************************

              RS232C.h

*********************************************/

#if !defined(RS_BUFSIZE)
#define RS_BUFSIZE  512
#endif

#define CBIT7       2
#define CBIT8       3

#define P_NONE      0
#define P_EVEN      1
#define P_ODD       3

#define SBIT1       1
#define SBIT15      2
#define SBIT2       3

#define INV_BAUD    -1
#define INV_BITS    -2


int rs_init(int ,int ,int ,int);
void rs_release(void );
int rs_getch(void );
void rs_putch(int c);
```

プログラム14 ★terminal.c★

```
/*********************************************

              Terminal Program

    Copyright (c) Naoshi Kimura    1988

*********************************************/

#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <io.h>

#include "rs232c.h"
#include "terminal.h"

static int redirected = 0;

void terminal(void)
{
    int c;

    while (1) {
        if (ioctl_isready(STDIN)) {
            c = readch(STDIN);
            if (c == '_' - '@')
                break;
            rs_putch(c);
        }
        if ((c = rs_getch()) != -1) {
            writech(STDERR, c);
            if (redirected)
                writech(STDOUT, c);
        }
    }
}

int main(int argc, char **argv)
{
    char *p;
    int baud = 1200;
    char *pbitstr = "n";
    int cbits = 8;
    char *sbitstr = "1";
    int sbits, pbit;
    int bfsave;
    int error = 0;

    while (*(p = *++argv) == '-') {
        switch (*++p) {
        case 'b':
            baud = atoi(++p);
            break;
        case 'c':
            cbits = atoi(++p);
            break;
        case 's':
            sbitstr = ++p;
            break;
        case 'p':
            pbitstr = ++p;
            break;
        default:
            cputs("Unknown option:");
            cputs(*argv);
```

```
		cputs("¥r¥n");
		return 4;
	}
}

/* check BAUDRATIO parameter */
switch (baud) {
case 75:
case 150:
case 300:
case 600:
case 1200:
case 2400:
case 4800:
case 9600:
case 19200:
	break;
default:
	cputs("No such baud ratio.");
	cputs(" -b{300|600|1200|2400|4800|9600|19200}¥r¥n");
	error = 1;
}

/* check character bit parameter */
switch (cbits) {
case 7:
	cbits = CBIT7;
	break;
case 8:
	cbits = CBIT8;
	break;
default:
	cputs("No such character bits.");
	cputs(" -c{7|8}¥r¥n");
	error = 1;
}

/* check STOP bit parameter */
if (strcmp(sbitstr, "1") == 0)
	sbits = SBIT1;
else if (strcmp(sbitstr, "1.5") == 0)
	sbits = SBIT15;
else if (strcmp(sbitstr, "2") == 0)
	sbits = SBIT2;
else {
	cputs("No such stopbits.")+
	cputs(" -s{1|1.5|2}¥r¥n");
	error = 1;
}

/* check PARITY bit parameter */
if (strcmp(pbitstr, "n") == 0)
	pbit = P_NONE;
else if (strcmp(pbitstr, "e") == 0)
	pbit = P_EVEN;
else if (strcmp(pbitstr, "o") == 0)
	pbit = P_ODD;
else {
	cputs("No such parity bit.")+
	cputs(" -p{n|e|o}¥r¥n");
	error = 1;
}
if (error)
	return 2;

redirected = isatty(STDOUT) == 0;
```

```
	/* initialize RS-232C port */
	if (rs_init(baud, cbits, sbits, pbit) != 0) {
		cputs("Cannot initialize RS-232C port¥r¥n");
		return 2;
	}

	bfsave = getbreakflag();/* save BREAK flag state */
	setbreakflag(0);        /* BREAK off */
	ioctl_raw(STDIN, 1);    /* raw mode */
	terminal();             /* terminal routine */
	rs_release();           /* reset RS-232C */
	ioctl_raw(STDIN, 0);    /* cooked mode */
	setbreakflag(bfsave);   /* resume BREAK flag */

	return 0;

}
```

プログラム15 ★rsio.c★

```
/*********************************************

          RSIO.c for Terminal Program

    Copyright (c) Naoshi Kimura   1988

*********************************************/

/*
 * rsio.c
 * RS-232C port I/O routines
 */

#include <conio.h>
#include <dos.h>
#include "rs232c.h"

#define PORT_SIO_MODE   0x32
#define PORT_SIO_DATA   0x30
#define PORT_SIO_STAT   0x32
#define PORT_PIC_OCW1   0x02
#define PORT_PIC_OCW2   0x00
#define PORT_MASKSET    0x37
#define PORT_TIMER_MODE 0x77
#define PORT_TIMER_CH2  0x75

#define PIC_EOI     0x20
#define PIC_MASK    0x10

static void (interrupt far *orgint)();
static int bufrp;
static int bufwp;
static int count;
static char buffer[RS_BUFSIZE];

#pragma intrinsic(inp, outp, _enable, _disable)

static void interrupt far rsintr(void)
{
    _enable();
    if (bufwp >= RS_BUFSIZE)
        bufwp = 0;
    buffer[bufwp++] = (char)inp(PORT_SIO_DATA);
    if (++count > RS_BUFSIZE)
        count = RS_BUFSIZE;
    _disable();
    outp(PORT_PIC_OCW2, PIC_EOI);
}

struct btab {
    int baud;
    int clock10;
    int clock8;
};

static struct btab baud_tab[10] = {
    19200, 8,       -1,
    9600,  16,      13,
    4800,  32,      26,
    2400,  64,      52,
    1200,  128,     104,
    600,   256,     208,
    300,   512,     416,
    150,   1024,    832,
```

```
	75,		2048,	1664
	-1,		-1,		-1
};

static void wait(void)
{
	/* */
}

static void sio_portinit(int mode, int command)
{
	command &= ~0x40;
	outp(PORT_SIO_MODE, command);
	wait();
	outp(PORT_SIO_MODE, command | 0x40);
	wait();
	outp(PORT_SIO_MODE, mode);
	wait();
	outp(PORT_SIO_MODE, command);
}


static void speed_portinit(int timer)
{
	outp(PORT_TIMER_MODE, 0xb4);
	outp(PORT_TIMER_CH2, timer & 0xff);
	outp(PORT_TIMER_CH2, (timer >> 8) & 0xff);
}


static int cpuclock(void)
{
	return inp(0x42) & 0x20;
}


int rs_init(int baud, int cbit, int sbit, int pbit)
{
	struct btab *bt = baud_tab;
	int tc;

	while (bt->baud != baud) {
		if ((bt++)->baud == -1)
			return INV_BAUD;
	}
	if ((tc = cpuclock() ? bt->clock8 : bt->clock10) == -1)
		return INV_BAUD;
	if ((cbit < CBIT7 || cbit > CBIT8) ||
		(sbit < SBIT1 || sbit > SBIT2) ||
		(pbit != P_NONE && pbit != P_EVEN && pbit != P_ODD))
		return INV_BITS;
	orgint = _dos_getvect(0x0c);
	_disable();
	sio_portinit((sbit << 6) | (pbit << 4) | (cbit << 2) | 0x02, 0x37);
	speed_portinit(tc);
	outp(PORT_PIC_OCW1, inp(PORT_PIC_OCW1) & ~PIC_MASK);
	outp(PORT_MASKSET, 1);
	_dos_setvect(0x0c, rsintr);
	_enable();
	bufrp = bufwp = count = 0;
	return 0;
}

void rs_release(void)
{
	_disable();
	outp(PORT_PIC_OCW1, inp(PORT_PIC_OCW1) | PIC_MASK);
```

```
    outp(PORT_MASKSET, 0);
    _dos_setvect(0x0c, orgint);
    _enable();
}

int rs_getch(void)
{
    register int c, bufp;

    if (count == 0)
        return -1;
    bufp = bufrp;
    if (bufp >= RS_BUFSIZE)
        bufp = 0;
    c = buffer[bufp];
    bufrp = ++bufp;
    --count;
    return c;
}

void rs_putch(int c)
{
    while ((inp(PORT_SIO_STAT) & 1) == 0);
    outp(PORT_SIO_DATA, c);
}
```

プログラム16　★dosfunc.c★

```
/*********************************************

        Dos Func.c for Terminal Program

    Copyright (c) Naoshi Kimura   1988

*********************************************/

#include <dos.h>

int ioctl_raw(int handle, int flag)
{
    union REGS reg;

    reg.x.bx = handle;
    reg.x.ax = 0x4400;
    intdos(&reg, &reg);       /* get IOCTL data */
    if (reg.x.cflag)
        return -1;
    if ((reg.x.dx & 0x0080) == 0)
        return -1;            /* handle is not a device */
    reg.h.dh = 0;
    reg.h.dl &= 0xdf;         /* mask bits */
    reg.h.dl |= flag ? 0x20 : 0x00;
    reg.x.ax = 0x4401;
    intdos(&reg, &reg);
    if (reg.x.cflag)
        return -1;
    return 0;
}

int ioctl_isready(int handle)
{
    union REGS reg;

    reg.x.bx = handle;
    reg.x.ax = 0x4406;
    intdos(&reg, &reg);
    return reg.x.cflag ? -1 : reg.x.ax != 0;
}


int readch(int handle)
{
    int c;
    unsigned bytes;

    _dos_read(handle, (void far *)&c, 1, &bytes);
    return c & 0xff;
}

void writech(int handle, int c)
{
    unsigned bytes;
    _dos_write(handle, (void far *)&c, 1, &bytes);
}

int getbreakflag(void)
{
    union REGS reg;

    reg.x.ax = 0x3300;
    intdos(&reg, &reg);
    return reg.h.dl;
}

void setbreakflag(int flag)
{
    union REGS reg;

    reg.x.ax = 0x3301;
    reg.h.dl = (unsigned char)(flag != 0);
    intdos(&reg, &reg);
}
```

3.8 ソートプログラム

MS-DOSコマンドにはSORTという外部コマンドがあります、このコマンドはファイルを読み込み、昇順・降順にソートして出力するプログラムです。ここでは、同じ様な動作を行うことのできるssort.cというサンプルプログラムを例にとって、メモリアロケーション関数の使い方、ソート関数の使い方を紹介しましょう。

3.8.1 ssort.cの使い方

標準入力から読み取ったテキストファイルをソートし、標準出力に出力します。

```
ssort < FILE1 > FILE2
type FILE1 | ssort > FILE2
```

この2例はどちらも同じ動作をします。

スモールモデルではあまり大きなファイルが扱えませんので、コンパイルの際にはコンパクトモデルを使用するようにしてください。扱えるファイルの大きさは、システムの残りメモリ量やファイルの行数により増減します。

3.8.2 メモリアロケーション関数の概要

メモリアロケーション関数は、プログラムで使ってもよいメモリを「合法的に」取得する関数です。コンピュータのメモリはDOSによって管理され、空いているから(空いているように見えるから)といって勝手に使ってよいものではありません。勝手にメモリを使うと、DOSの管理領域を破壊したり、他のプログラムが使用している領域と重なったりして不都合が生じます。メモリが必要になったら、これらの関数を使用してメモリ確保を行わなければなりません。

標準的なメモリアロケーション関数にはふたつの種類があります。スタック上に領域を確保する関数と、データ領域からメモリを確保する関数です。

スタック上からメモリを確保する関数にはalloca関数があります。この関数は与えられたサイズ分の領域をスタック上に確保します。そのため、alloca関数を呼び出した関数が終了すると領域が捨てられます。関数内だけで有効な可変長のバッファを取りたいときに有用でしょう。関数が終了すると自動的に解放されますので、領域フリーの作業は不要です。ただし、データ領域に比べスタック領域は狭いので、大きな領域を確保した場合、スタックオーバーフローをひき起こす可能性があります。この関数は小さな領域を確保するのに適しています。

データ領域にメモリを確保する関数にはmalloc関数、calloc関数があります。malloc関数では、指定されたバイト数分のメモリをデータ領域から確保して、そのポインタを返します。alloca関数と違い、領域を解放するかプログラムが終了しないかぎり、mallocで確保された領域は保証されます。malloc関数で確保された領域を解放する関数がfree関数です。malloc関数で確保した領域が不要になったならば、free関数で解放しておくべきです。calloc関数は高機能なmalloc関数と考えてください。malloc関数では単純にバイト数を与え領域を確保するのに対し、calloc関数では指定されたバイト数の領域を指定された数だけ確保して、要素をすべて0に初期化します。確保された領域の解放は、mallocと同じくfree関数で行います。

メモリアロケーション関数に分類されない関数でも、mallocを使用してメモリ確保を行っている関数もあります。たとえば、strdup関数では引数の文字列の複製を作り出しますが、複製を格納する領域の確保はmalloc関数を使用します。カレントディレクトリを取得するgetcwd関数では、格納バッファにNULLを指定すると領域確保を行った上で、カレントディレクトリを表す文字列へのポインタを返します。

その他、便利なメモリアロケーション関数として_fmalloc関数があります。この関数はfar領域から領域を確保する関数です。スモールモデル、ミディアムモデルのmalloc関数はnear領域からメモリを確保するため、確保するメモリ領域の合計が64KBを越えることは不可能です。大量のメモリを確保しなければならない場合には_fmalloc関数を使用します。_fmalloc関数を用いてもひとつの領域で64KBを越える領域は確保できません。そのような場合には、halloc関数を用います。_fmalloc関数の返す値はfarポインタとなり、halloc関数の返す値はhugeポインタとなります。far、hugeポインタは、

スモールモデル・ミディアムモデルの標準的な関数では直接取り扱うことはできません。FP_SEG、FP_OFFマクロや、movedata関数、far、huge型ポインタ等を使ってプログラミングする必要があります。

この他、DOSレベルでのメモリアロケートを行う、_dos_allocmem関数等が用意されています。リファレンスを参照してください。

●メモリアロケート関数の使い方

このプログラムでは、メモリアロケート関数として、

```
malloc()
calloc()
strdup()
```

関数を使用しています。

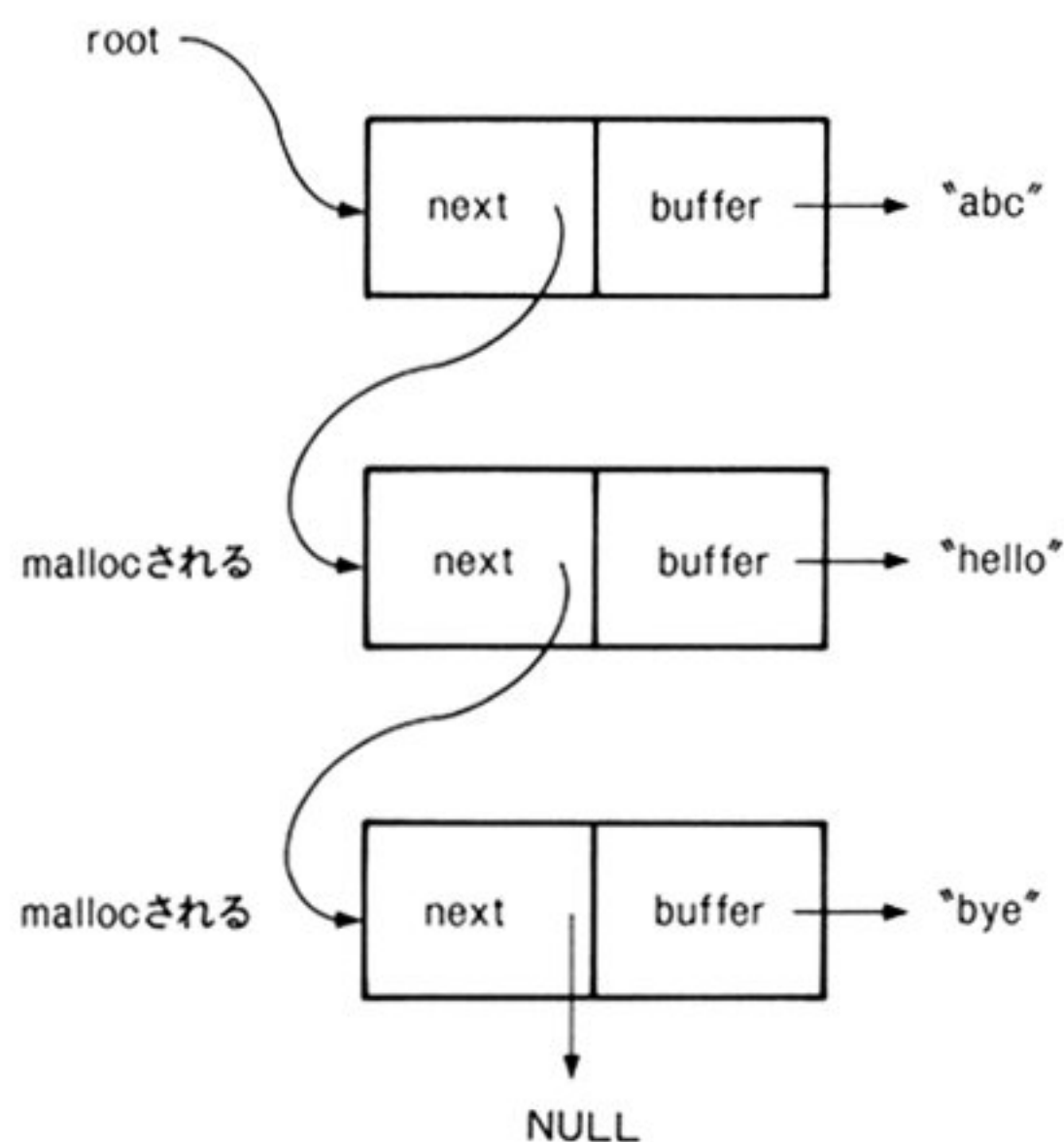
main関数では、まずファイルのデータをすべてメモリ上に読み込みます。構造体listはリスト形式です。ファイルから1行読み込まれるごとにmalloc関数によって領域を確保し、読み込まれた文字列の情報を格納します。読み込まれた内容を格納するには

root

next | buffer → "abc"

mallocされる → next | buffer → "hello"

mallocされる → next | buffer → "bye"

NULL

リスト構造体の連結例
入力文字列として
"abc"
"hello"
"bye"
が与えられた場合

配列でも構いません。しかし、配列を用いると一定の行数しか読み込めなくなってしまいます。メモリアロケート関数により、1行読み込まれるごとに必要な量だけメモリを確保していけば、ファイルの行数に応じて確保する量を増減させることができます。構造体listは、次に連結する構造体listへのポインタを自分自身の中にstruct list *next;として持っています。構造体listは1行読み込まれるごとにひとつずつ連結されて、ファイルすべての情報を格納します。

読み込まれた1行の文字列は、strdup関数で複製され保存されます。構造体listに格納される情報は複製された文字列へのポインタです。この文字列は後でソートをする際に比較のために用いられます。

読み込みが終了すると、リストの最終位置を示すためにnextの要素にNULLを代入しておきます。これはリストの終端を示すもので、後で読み出すときにチェックされます。

次にソートのための配列を作成します。配列は文字列へのポインタ配列で、calloc関数により配列の領域が確保されます。変数lには読み込まれた行数が格納されていますので、確保する要素数はl、ひとつの要素のサイズはキャラクタのポインタのサイズとなります。

確保されたソート用の配列に、構造体listに保存されている文字列へのポインタをコピーします。文字列へのポインタは、構造体listを先頭から次々にたどっていくことによって得ることができます。

その後qsort関数によりソートが行われ、結果を出力します。qsort関数はソーティングを行う関数で、ソートを行う要素の先頭へのポインタ、要素数、要素のサイズ、比較を行う関数を与えれば自動的にソーティングを行ってくれます。

プログラム17　★ssort.c★

```
/*********************************************
                 Sorting Program

     Copyright (c) Naoshi Kimura    1988

*********************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct list {
    char *buffer;
    struct list *next;
};

struct list root;

int compfunc(char **p1, char **p2)
{
    return strcmp(*p1, *p2);
}

int main(int argc, char **argv)
{
    char buffer[1000];
    char **strings, **sp;
    struct list *tp;
    int i, l;

    tp = &root;
    l = 0;
    while (gets(buffer) != NULL) {
        if ((tp->next = malloc(sizeof(struct list))) == NULL) {
            fputs("no memory¥n", stderr);
            return 2;
        }
        tp = tp->next;
        if ((tp->buffer = strdup(buffer)) == NULL) {
            fputs("no memory¥n", stderr);
            return 2;
        }
        l++;
    }
    tp->next = NULL;

    tp = &root;
    if ((strings = calloc(l, sizeof(char *))) == NULL) {
        fputs("no memory for sort table¥n", stderr);
        return 2;
    }
    sp = strings;
    while (tp = tp->next, tp != NULL)
        *sp++ = tp->buffer;
    qsort(strings, l, sizeof(char *), compfunc);
    sp = strings;
    for (i = 0; i < l; i++)
        puts(*sp++);
}
```

II

リファレンス

The References

【リファレンスの見方】

rename ❶ ver 5# ❾

■書式

#include 〈io.h〉 ❷
#include 〈stdio.h〉

int rename (oldname, newname); ❸

❹

char ・oldname; 前の名前
char ・newname; 新しい名前

■戻り値 ❺

成功なら0、エラーの場合は0以外の値を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	newnameによって指定したファイルが既に存在するか、作成できない。 またはoldnameがディレクトリであって、newnameが異なるパスを指定している。
ENOENT	oldnameで指定したファイルまたはパス名が存在しない。
EXDEV	ファイルを異なるデバイスへ移そうとした。

■機能 ❻

oldnameで指定したファイルまたはディレクトリをnewnameで指定した名前に変更します。
newnameで指定したファイルまたはディレクトリ名は存在しないものでなくてはなりません。

■ポイント ❼

引数newnameで別のパス名を指定することにより、ファイルを他のディレクトリに移動することが可能です。ただし異なるデバイスへの移動はできません。またディレクトリの移動はできません。

■例

簡単なrenameコマンドです。 ❽

```
#include <io.h>
#include <stdio.h>

void main(ac, av)
int ac;
char **av;
{
        if (ac != 3)
                exit(1);
        if (rename(av[1], av[2]) != 0) {
                printf("cannot rename %s to %s¥n", av[1], av[2]);
                exit(1);
        }
        exit(0);
}
```

❶ 関数名・バージョン

関数名そのものと、初めてライブラリとして供給されたバージョンです。無記入はバージョン3から存在するものです。ver 4はバージョン4から、ver 5はバージョン5から新たに加わった関数です。ver 5♯はバージョン5になって、その機能に変更等があったことを意味します。また、この欄にはANSI、UNIX等の準拠基準が記されることもあります。

❷ 書式・インクルードファイル

その関数を使用するために、宣言しておかなければならないインクルードファイル名です。

❸ 書式・関数名

関数の型と関数名です。

❹ 書式・引数

関数に引数があるときには、その引数の型を明示してあります。

❺ 戻り値

関数には、一般に戻り値があります。戻り値の型は関数の型そのものですが、その意味を書いてあります。戻り値以外のグローバル変数に、実行結果(主としてエラーの種類)などがある場合には、その内容を記述してあります。

❻ 機能

何をする関数であるか、またその動作について記述しています。

❼ ポイント

類似する関数、バージョンによる動作の違い、使用に関する注意をまとめました。特に必要がないと判断したものについては、記述はありません。

❽ 例

使用例のプログラムです。実際の使われ方がわかります。動作はすべて確認済みです。特に必要がないと判断したものについては記述はありません。

❾ 関数グループ名

その関数(あるいはマクロ)が属するグループです。

【リファレンスの構成】

MicrosoftCのリファレンスマニュアルは2冊構成になっています。リファレンス1ではインクルードファイルや関数の機能によるグループに分けて解説してあります。しかし、関数ひとつひとつについての説明であるリファレンス2では、単純にアルファベット順の並びになっています。

実際にユーザーが使用する場合、グループ分けしてあるリファレンスの方が使いやすいと思われます。しかしもちろん、アルファベット順で調べたいこともあります。そこで本書では、アルファベット順については別項を設けることとして、機能別にグループ分けした記述をメインとすることにしました。

各グループは主にインクルードファイルごとの構成になりますが、実際には複数のインクルードファイルにまたがっている場合があります。これはANSIに準拠した構成と、UNIXのSystem Vに準拠した構成による違いが原因です。また、stdlibなどのように雑多な関数が定義されているインクルードファイルもあります。そこで本書では、インクルードファイルにあまりこだわらず、機能別にグループ分けすることにしました。結果として、MicrosoftCマニュアル(リファレンス)の分け方とほぼ同様の構成となっています。ただし、グループ内での分け方は類似関数をまとめるようにしました。機能別に分けたグループは以下の通りです。

ストリーム入出力関数
コンソール入出力関数

低レベル入出力関数

ファイル操作関数
システムコール関数
プロセス制御関数
ディレクトリ操作関数

バッファ操作関数
メモリ割り当て関数

データ変換関数
文字の分類・変換関数
文字列操作関数

ソート・サーチ関数
時間関数
数値演算関数
可変長の引数の並び
スタンダード関数その他

日本語文字の分類・変換関数
日本語文字列操作関数

グラフィック関数

【各グループの機能の概略】

◆ストリーム入出力関数

バッファリングされる高水準入出力関数グループです。各ストリームなどについての詳細は第1部3章を参照してください。

◆コンソール入出力関数

ストリーム入出力の中でも、標準入出力専用の関数グループです。もちろんリダイレクトできます。

◆低レベル入出力関数

バッファリングされない低水準関数グループです。バッファはプログラマが用意しないといけません。

◆ファイル操作関数

ファイルの各種操作をするための関数グループです。

◆システムコール関数

MS-DOSのシステムコールや8086の割り込みを使うための関数グループです。バージョン5.1から、代表的システムコールはあらかじめ用意されるようになりました。

◆プロセス制御関数

子プロセスやMS-DOSのコマンドを実行するための関数グループです。

◆ディレクトリ操作関数

ディレクトリの作成や削除、移動などを行うための関数グループです。

◆バッファ操作関数

メモリ内容の移動や比較を行うための関数グループです。

◆メモリ割り当て関数

メモリの割り当て、解放を行うための関数グループです。

◆データ変換関数

データの形式を変換する関数グループです。

◆文字の分類・変換関数

文字をキャラクタコードによって分類・変換する関数グループです。

◆文字列操作関数

文字列のコピー、比較、切り出しなどを行う関数グループです。

◆ソート・サーチ関数

配列のソート、サーチを行う関数グループです。

◆時間関数

システム時間、日付に関する操作を行う関数グループです。

◆数値演算関数

数値を操作する数学的関数のグループです。

◆可変長の引数の並び

引数の数が固定でない関数を用いるときに使う関数グループです。

◆スタンダード関数その他

機能的にグループ分けされない、その他の標準的関数グループです。

◆日本語文字の分類・変換関数

シフトJISコードによる日本語文字の分類、変換をする関数グループです。

◆日本語文字列操作関数

シフトJISコードを含んだ文字列の操作をする関数グループです。

◆グラフィック関数

グラフィック画面を操作する関数グループです。バージョン5.1からサポートされました。この関数は機種依存ですから、使用には注意が必要です。

【各関数の機能の概略】

関数の詳しい機能等についてはリファレンスを参照していただきますが、ここではその概略をまとめます。バージョン4および5からライブラリに加えられたもの、およびバージョンによって機能が多少異なるものもありますのでご注意ください。これらについては関数名の前に〝4、5、#″の記号を付けて明示しました。

◆バージョン4から用意された関数　4
◆バージョン5から用意された関数　5
◆バージョン5から変更のあった関数　#　(詳細はリファレンス参照)

■ストリーム入出力関数

STDIO.H:

	fopen	ストリームをオープンする。
	freopen	ストリームを再割り当てする。
#	fclose	ストリームをクローズする。
#	fcloseall	オープンされているすべてのストリームをクローズする。
	fdopen	ファイルハンドルを使ってストリームをオープンする。
	feof (macro)	ストリームのEOFを調べる
	ferror (macro)	ストリームのエラーを調べる。
	fflush	ストリームをフラッシュする。
	flushall	オープンされているすべてのストリームをフラッシュする。
#	fscanf	ストリームからフォーマット形式でデータを読み込む。
	scanf	stdinからフォーマット形式でデータを読み込む。
#	sscanf	文字列からフォーマット形式でデータを読み込む。
	fgetc	ストリームから1文字読み込む。
	fgetchar	stdinから1文字読み込む。
	fgets	ストリームから1行読み込む。
	fread	ストリームから固定長データを読み込む。
	getc (macro)	ストリームから1文字読み込む。
	getchar (macro)	stdinから1文字読み込む。
	gets	stdinから1行読み込む。
	getw	ストリームからint型のデータを読み込む。
	ungetc	バッファに文字を戻す。
#	fprintf	ストリームにフォーマット形式でデータを書き込む。
	printf	stdoutにフォーマット形式でデータを書き込む。
#	sprintf	文字列バッファにフォーマット形式でデータを書き込む。
	fputc	ストリームへ1文字出力する。
	fputchar	stdoutへ1文字出力する。
	fputs	ストリームへ1行出力する。
	fwrite	ストリームへ固定長データを出力する。
	putc (macro)	ストリームへ1文字出力する
	putchar (macro)	stdoutへ1文字出力する。
	puts	stdoutへ1行出力する。
	putw	ストリームへint型データを書き込む。

	fseek	ファイルポインタを移動する。
	ftell	現在のファイルポインタの位置を得る。
5	fgetpos	現在のファイルポインタインジケータを得る。
5	fsetpos	ファイルポインタインジケータを設定する。
	rewind	ファイルポインタを先頭に移す。
	clearerr	エラーインジケータをクリアする。
	fileno (macro)	ストリームのファイルハンドルを得る。
4	rmtmp	テンポラリファイルを消去する。
	setbuf	ストリームのバッファを変更する。
4&#	setvbuf	ストリームのバッファとサイズを変更する。
4	tempnam	テンポラリファイル名を作成して名前を返す。
4&#	tmpfile	テンポラリファイルを作成してそのポインタを返す。
4	tmpnam	テンポラリファイル名を作成して名前のポインタを返す。
4	vfprintf	標準出力に引数並びをフォーマット形式で出力する。
4	vprintf	stdoutに引数並びをフォーマット形式で出力する。
4	vsprintf	文字列領域に引数並びをフォーマット形式で出力する。

■コンソール入出力関数

CONIO.H:

	kbhit	コンソールからのキー入力の有無を調べる。
	getch	コンソールからの1文字入力(エコーバックなし)。
	getche	コンソールからの1文字入力(エコーバックあり)。
	cgets	コンソールからの1行入力。
#	cscanf	コンソールからのフォーマット形式入力。
#	putch	コンソールへの1文字出力。
	ungetch	コンソールへ1文字戻す。
#	cputs	コンソールへの1行出力。
#	cprintf	コンソールへのフォーマット形式の出力。
	inp	入力ポートから1バイト読み込む。
5	inpw	入力ポートから1ワード読み込む。
	outp	出力ポートに1バイト出力する。
5	outpw	出力ポートに1ワード出力する。

■低レベル入出力関数

IO.H:

open	FCNTL.H	ファイルをオープンする。
sopen	SHARE.H	ファイルを共有モードでオープンする。
close		ファイルをクローズする。
creat		ファイルを作成する。
write	FCNTL.H	ファイルに指定バイト数書き込む。
read		ファイルから指定バイト数読み込む。
lseek		ファイルポインタを移動させる。
tell		ファイルポインタの位置を調べる。
eof		ファイルの終了であるか調べる。
dup		オープンしているファイルハンドルを複製する。
dup2		ファイルハンドルを再割り当てする。

■ファイル操作関数

IO.H:

	access		指定ファイルの存在とモードを調べる。
	chmod		ファイルの属性を変更する。
	chsize		ファイルの大きさを変更する。
	filelength		ファイルの長さをバイト単位で返す。
	fstat		ファイルに関する情報を得る。
	isatty		デバイスが文字型であるか調べる。
	locking		ファイルのロック・アンロックを設定する。
5	_makepath		パス名をパーツから作成する。
	mktemp		テンポラリファイル名を作成しファイル名へのポインタを返す。
4	remove	STDIO.H	ファイルを削除する。
	rename	STDIO.H	ファイル名・ディレクトリ名を変更する。
	setmode		ファイルのモードを変更する。
5	_splitpath		パス名をパーツに分解する。
	stat		ファイル情報を獲得する。
	umask		ファイルアクセス許可設定を変更する。
	unlink	STDIO.H	ファイルを削除する。

■システムコール関数

DOS.H:

	bdos	MS-DOSのシステムコールを実行する。
5	_chain_intr	割り込みハンドラから別の割り込みハンドラを起動する。
5	_disable	ハードウェア割り込みを禁止する。
5	_dos_allocmem	メモリブロックを割り当てる。
5	_dos_close	ファイルクローズを行う。
5	_dos_creat	新しいファイルをオープンしファイルハンドルを返す。
5	_dos_creatnew	新しいファイルをオープンしファイルハンドルを返す。
	dosexterr	拡張エラー情報をバッファに返す。
5	_dos_findfirst	最初に一致するディレクトリエントリを検索する。
5	_dos_findnext	次のディレクトリエントリを検索する。
5	_dos_freemem	dos_allocmem()により割り当てられたメモリブロックを解放する。
5	_dos_getdate	システムクロックから現在の日付を得る。
5	_dos_getdiskfree	指定されたディスクの情報を得る。
5	_dos_getdrive	カレントドライブ番号を取得する。
5	_dos_getfileattr	pathで指定されたファイルディレクトリの属性を得る。
5	_dos_getftime	オープンされているファイルの最終更新日時を得る。
5	_dos_gettime	現在時刻を取得する。
5	_dos_getvect	割り込みベクタvectの内容を得る。
5	_dos_keep	プログラムを常駐終了させる。
5	_dos_open	ファイルをオープンしファイルハンドルを返す。
5	_dos_read	ファイルから指定バイト読み込み、バッファに格納する。
5	_dos_setblock	_dos_alocmemで割り当てたメモリブロックの大きさを変更する。
5	_dos_setdate	システム日付を変更する。
5	_dos_setdrive	カレントドライブを変更する。
5	_dos_setfileattr	指定ファイルの属性を変更する。

5	_dos_setftime	オープンされているファイルの最終更新日時を変更する。
5	_dos_settime	現在のシステム時刻を変更する。
5	_dos_setvect	割り込みベクタ番号の処理アドレスを変更する。
5	_dos_write	指定ファイルに書き込む。
5	_enable	ハードウェア割り込みを許可する。
	FP_SEG (macro)	ポインタのセグメント部分の値を得る。
	FP_OFF (macro)	ポインタのオフセット部分の値を得る。
5	_harderr	INT 24Hのハンドラを設定する。
5	_hardresume	ハードウェアエラーハンドル関数からMS-DOSに戻る。
5	_hardretn	ハードウェアエラーハンドル関数からアプリケーションに戻る。
	int86	8086プロセッサの割り込みを実行する。
	int86x	8086プロセッサの割り込みを実行する。(セグメントレジスタの値を指定)
	intdos	MS-DOSシステムコールを起動する。
	intdosx	MS-DOSシステムコールを起動する。(セグメントレジスタの値を指定)
	segread	セグメントレジスタの値を得る。

■プロセス制御関数

STDLIB.H

	atexit	正常終了したときに起動する関数を指定する。

PROCESS.H:

#	abort STDLIB.H	stderrにメッセージを出力し異常終了する。
5	cwait	指定した子プロセスが終了するまで呼び出したプロセスを停止する。(OS/2)
	execl	引数並びを指定して子プロセスを実行する。
	execle	引数並びと動作環境を指定して子プロセスを実行する。
	execlp	引数並びを指定して子プロセスを実行する。環境変数PATHを参照する。
4	execlpe	引数並びと動作環境を指定して子プロセスを実行する。環境変数PATHを参照する。
	execv	引数の配列を指定して子プロセスを実行する。
	execve	引数の配列と動作環境を指定して子プロセスを実行する。
	execvp	引数の配列を指定して子プロセスを実行する。環境変数PATHを参照する。
4	execvpe	引数の配列と動作環境を指定して子プロセスを実行する。環境変数PATHを参照する。
	exit STDLIB.H	ストリームバッファをフラッシュしてプロセスを終了する。
	_exit STDLIB.H	ストリームバッファをフラッシュせずプロセスを終了する。
	getpid	現在のプロセスIDを返す。
4&#	onexit	正常終了したときに起動する関数を指定する。
	spawnl	引数並びを指定して子プロセスを実行する。
	spawnle	引数並びと動作環境を指定して子プロセスを実行する。
	spawnlp	引数並びを指定して子プロセスを実行する。環境変数PATHを参照する。
4	spawnlpe	引数並びと動作環境を指定して子プロセスを実行する。環境変数PATHを参照する。
	spawnv	引数の配列を指定して子プロセスを実行する。
	spawnve	引数の配列と動作環境を指定して子プロセスを実行する。
	spawnvp	引数の配列を指定して子プロセスを実行する。環境変数PATHを参照する。
4	spawnvpe	引数の配列と動作環境を指定して子プロセスを実行する。環境変数PATHを参照する。

system STDLIB.H MS-DOSコマンドの実行。
5 wait 子プロセスが終了するまで呼び出したプロセスを停止する。(OS/2)

SIGNAL.H:

raise 実行中のプログラムにSIGNALを送る。
signal 割り込みシグナルの取り扱いを指定する。

■ディレクトリ操作関数

DIRECT.H:

chdir カレントディレクトリの移動。
getcwd カレントディレクトリの名前を得る。
mkdir 指定したディレクトリを作成する。
rmdir 指定したディレクトリを削除する。

■バッファ操作関数

MEMORY.H:

memccpy STRING.H メモリを指定したバイト数または指定文字までコピーする。
memcpy STRING.H メモリの指定したバイト数をコピーする。
memchr STRING.H メモリ中から指定文字を探す。
memcmp STRING.H メモリを比較する。
4 memicmp STRING.H メモリを大文字小文字を区別せずに比較する。
memset STRING.H メモリを指定文字で充塡する。
movedata STRING.H セグメントとオフセットの指定によって文字をコピーする。

STRING.H

5 memmove メモリをコピーする。領域の重複を許す。

■メモリ割り当て関数

MALLOC.H:

4 alloca スタック領域からメモリを割り当てる。
4 _expand メモリのブロックサイズを変更する。
4 _ffree _fmallocで割り当てたメモリブロックを解放する。
5 _fheapchk far heapの整合性をチェックする。
5 _fheapset far heapの整合性をチェックし、指定した値で充塡する。
5 _fheapwalk far heapのエントリ情報を_headinfo構造体に格納する。
4 _fmalloc デフォルト以外のデータセグメントからデータブロックを割り当てる。
4 _fmsize _fmallocで割り当てられたサイズを調べる。
4 _freect 動的に割り当て可能なメモリを調べる。
4 halloc huge配列を割り当てる。
4 hfree hallocで割り当てたhuge配列を解放する。
5 _heapchk (macro) heapの整合性をチェックする。
5 _heapset (macro) heapの整合性をチェックし、指定した値で充塡する。
5 _heapwalk (macro) heapのエントリ情報を_headinfo構造体に格納する。
4 _memavl デフォルトDSから割り当てられる最大メモリサイズを調べる。
5 _memmax near heapから連続して割り当てられる最大メモリブロックを調べる。
4&# _msize calloc、malloc、reallocで割り当てられたメモリブロックサイズを調べる。
4 _nfree _nmallocで割り当てられたメモリブロックを解放する。
5 _nheapchk near heapの整合性をチェックする。
5 _nheapset near heapの整合性をチェックし、指定した値で充塡する。
5 _nheapwalk near heapのエントリ情報を_headinfo構造体に格納する。

4 _nmalloc　デフォルトDSからメモリブロックを割り当てる。
4 _nmsize　_nmallocで割り当てられたメモリブロックサイズを調べる。
calloc STDLIB.H　配列を割り当てる。
free STDLIB.H　メモリブロックを解放する。
malloc STDLIB.H　メモリブロックを割り当てる。
realloc STDLIB.H　すでに割り当てられているメモリブロックの大きさを変更する。
sbrk　プロセスのブレーク値を再設定する。
4 stackavail　スタック領域から動的にメモリ割り当てできるサイズを調べる。

■データ変換関数

STDLIB.H:

atoi　stringをintに変換する。
atol　stringをlongに変換する。
atof MATH.H　stringをdoubleに変換する。
itoa　intをstringに変換する。
ltoa　longをstringに変換する。
ultoa　unsigned longをstringに変換する。
ecvt　doubleをstringに指定桁数だけ変換する。
fcvt　doubleをstringに指定小数点桁数だけ変換する。
gcvt　doubleをstringに変換する。
4 strtod　stringをdoubleに変換する。
4 strtol　stringをlongに変換する。
5 strtoul　stringをunsigned longに変換する。

■文字の分類・変換関数

CTYPE.H:

isalpha (macro)　英字かどうか判別する。
isupper (macro)　大文字かどうか判別する。
islower (macro)　小文字かどうか判別する。
isdigit (macro)　数字かどうか判別する。
isxdigit (macro)　16進数の数字かどうか判別する。
isspace (macro)　空白文字かどうか判別する。
ispunct (macro)　句読点かどうか判別する。
isalnum (macro)　英数字かどうか判別する。
isprint (macro)　表示可能な文字かどうか判別する。(0x20を含む)
isgraph (macro)　表示可能な文字かどうか判別する。
iscntrl (macro)　制御文字かどうか判別する。
isascii (macro)　アスキー文字かどうか判別する。
_toupper (macro)　大文字に変換する。(0x20減)
_tolower (macro)　小文字に変換する。(0x20加)
toupper (macro)　大文字に変換する。
tolower (macro)　小文字に変換する。
toascii (macro)　アスキー文字列に変換する。
iscsymf (macro)　下線文字(_)であるかどうか判別する。
iscsym (macro)　下線文字(_)であるかどうか判別する。

■文字列操作関数

STRING.H:

	strcat	文字列をcatする。
	strncat	文字列を指定文字だけcatする。
	strcmp	文字列を比較する。
	strcmpi	文字列を大文字小文字の区別なしに比較する。
4	stricmp	文字列を大文字小文字の区別なしに比較する。
4	strncmp	文字列を指定文字数だけ比較する。
	strncmpi (macro)	文字列を指定文字数だけ大文字小文字の区別なしに比較する。
4	strnicmp	文字列を指定文字数だけ大文字小文字の区別なしに比較する。
	strcpy	文字列をコピーする。
	strncpy	文字列を指定文字数だけコピーする。
	strdup	文字列を複製する。
	strlen	文字列の長さを調べる。
	strset	文字列を指定文字で充填する。
	strnset	文字列を指定文字数だけ指定文字で充填する。
	strcspn	文字列中に指定した文字が現れるまでのバイト数を調べる。
4&#	strerror	エラー番号に対応するエラーメッセージへのポインタを返す。
	_strerror	エラー番号に対応するエラーメッセージへのポインタを返す。
	strpbrk	指定した文字セットの目的文字列中での位置を調べる。
	strspn	指定した文字セット以外の目的文字列中での位置を調べる。
4	strstr	指定した文字列の別文字列中での位置を調べる。
	strtok	文字列をトークンで切り出す。
	strchr	指定文字の別文字列中での位置を前から調べる。
	strrchr	指定文字の別文字列中での位置を後ろから調べる。
	strrev	文字列並びを逆転させる。
	strlwr	文字列を小文字に変換する。
	strupr	文字列を大文字に変換する。

■ソート・サーチ関数

SEARCH.H:

4	lfind	リニアサーチを行う。
4	lsearch	リニアサーチを行う。
	bsearch STDLIB.H	バイナリサーチを行う。
	qsort STDLIB.H	クイックソートを行う。

■時間関数

TIME.H:

	asctime	tm構造体型の時間情報を文字列に変換する。
	time	現在のシステム時間を秒数で返す。
#	ctime	time_t型の時刻を文字列に変換する。
#	gmtime	time_t型の時刻をGMTを表わすtm型構造体に変換する。
#	localtime	time_t型の時刻を地方時を表わすtm型構造体に変換する。
4	difftime	ふたつの時刻の差を秒数で返す。
5	mktime	地方時をカレンダー値に変換する。
	tzset	環境変数TZによってグローバル変数をセットする。
5	clock	プロセスの実行に要したプロセッサ時間を調べる。

5	_strdate	現在の日付を文字列に変換する。
5	_strtime	現在の時刻を文字列に変換する。

TIMEB.H:

	ftime	システム時間を得る。

UTIME.H:

	utime	ファイル更新日時をセットする。

■数値演算関数

FLOAT.H:

4	_clear87	浮動小数点ステータスワードをクリアする。
4	_control87	浮動小数点コントロールワードを設定する。
4	_fpreset	浮動小数点に関する数値演算パッケージを初期化する。
4	_status87	浮動小数点ステータスワードを返す。

MATH.H:

	acos	アークコサインを返す。
	asin	アークサインを返す。
	atan	アークタンジェントを返す。
	atan2	アークタンジェントを返す。(引数が0のとき0を返す)
	cabs	複素数の絶対値を返す。
	ceil	double型の数を越えた最小の整数値を返す。
	cos	コサインを返す。
	cosh	ハイパボリックコサインを返す。
4	dieeetomsbin	IEEEフォーマットのdoubleをMSバイナリに変換する。
4	dmsbintoieee	MSバイナリフォーマットのdoubleをIEEEに変換する。
	exp	eのn乗を返す。
	fabs	絶対値を返す。
4	fieeetomsbin	IEEEフォーマットのfloatをMSバイナリに変換する。
	floor	double型の数を越えない最大の整数値を返す。
	fmod	float型の剰余を返す。
4	fmsbintoieee	MSバイナリフォーマットのfloatをIEEEに変換する。
	frexp	double型の値を仮数部と指数部に分解する。
	hypot	直角三角形の斜辺の長さを返す。
	j0,j1,jn,y0,y1,yn	ベッセル関数の値を返す。
	ldexp	2のx乗を返す。
#	log	自然対数を返す。
#	log10	底が10の対数を返す。
	matherr	数値演算中のエラーを処理する。
	modf	double型を整数部と分数部に分ける。
	pow	xのn乗を返す。
	sin	サインを返す。
	sinh	ハイパボリックサインを返す。
	sqrt	平方根を返す。
	tan	タンジェントを返す。
	tanh	ハイパボリックタンジェントを返す。

■可変長の引数の並び

STDARG.H:　(ANSI)

VARARGS.H: (UNIX)

	関数		説明
	va_start	(macro)	引数のポインタを可変長引数リストの先頭にセットする。
	va_arg	(macro)	現在の引数を返す。
	va_end	(macro)	現在の引数をリセットする。

■スタンダード関数その他

ASSERT.H:

	関数		説明
#	assert	(macro)	論理エラーをテストする。

SETJMP.H:

	関数		説明
	longjmp		setjmp関数で保存したスタック環境を復元し他の関数へジャンプする。
	setjmp		longjmpの実行時に復元するスタック環境を保存する。

STDLIB.H:

	関数		説明
	abs	MATH.H	絶対値を返す。
	labs	MATH.H	long型の整数値の絶対値を返す。
4	min		ふたつの数値を比較し小さい方の値を返す。
4	max		ふたつの数値を比較し大きな方の値を返す。
5	div		整数を除算し商と剰余をdiv_t構造体に格納する。
	getenv		環境変数の内容を得る。
5	ldiv		long型整数を除算し商と剰余をdiv_t構造体に格納する。
	putenv		指定した環境変数をセットする。
	rand		擬似乱数を返す。
	srand		擬似乱数の乱数系列を初期化する。
	swab		データの上位・下位バイトを交換する。
5	_lrotl		unsignd long型の値を左にローテートさせた値を返す。
5	_lrotr		unsignd long型の値を右にローテートさせた値を返す。
5	_rotl		unsignd int型の値を左にローテートさせた値を返す。
5	_rotr		unsignd int型の値を右にローテートさせた値を返す。
5	_searchenv		ファイルをサーチしフルパス名を得る。
	perror	STDIO.H	stderrにエラーメッセージを出力する。

■日本語文字の分類・変換関数

JCTYPE.H:

関数		説明
iskana	(macro)	半角カナかどうか判別する。
iskpun	(macro)	半角カナの句読点かどうか判別する。
iskmoji	(macro)	半角カナ文字かどうか判別する。
isalkana	(macro)	英文字カナ文字かどうか判別する。
ispnkana	(macro)	英カナ句読点かどうか判別する。
isalnmkana	(macro)	英数字カナ文字かどうか判別する。
isprkana	(macro)	表示文字かどうかを判別する。
isgrkana	(macro)	空白文字を含む表示可能文字かどうかを判別する。
iskanji	(macro)	全角文字コードの1バイト目かどうか判別する。
iskanji2	(macro)	全角文字コードの2バイト目かどうか判別する。

■日本語文字列操作関数

JSTRING.H:

btom	nバイト目までにある全角文字数を返す。
chkctype	文字の種類を調べる。
hantozen	半角文字を全角文字に変換する。
jisalpha	全角のアルファベットであるか判別する。
jisdigit	全角の数字かどうか判別する。
jishira	全角のひらがなかどうか判別する。
jiskata	全角のカタカナかどうか判別する。
jiskigou	全角の句読点かどうか判別する。
jisl0	全角のひらがな・カタカナ・英数字かどうか判別する。
jisl1	第1水準漢字かどうか判別する。
jisl2	第2水準漢字かどうか判別する。
jislower	全角の小文字かどうか判別する。
jisspace	全角の空白文字かどうか判別する。
jistojms	JIS漢字コードに変換する。
jisupper	全角の大文字かどうか判別する。
jiszen	全角文字かどうか判別する。
jmstojis	Shift-JISコードに変換した値を返す。
jstradv	指定文字数だけポインタを進める。
jstrchr	指定文字の別文字列中での位置を前から調べる。
jstrcmp	文字列を比較する。
jstrlen	文字列の長さを調べる。
jstrmatch	指定した文字セットの目的文字列中での位置を調べる。
jstrncat	文字列を指定文字だけcatする。
jstrncmp	文字列を指定文字数だけ比較する。
jstrncpy	文字列を指定文字数だけコピーする
jstrrchr	指定文字の別文字列中での位置を後ろから調べる。
jstrrev	文字列並びを逆転させる。
jstrskip	指定した文字セット以外の目的文字列中での位置を調べる。
jstrstr	指定した文字列の別文字列中での位置を調べる。
jstrtok	文字列をトークンで切り出す。
jtohira	全角のひらがなに変換する。
jtokata	全角のカタカナに変換する。
jtolower	全角小文字に変換する。
jtoupper	全角大文字に変換する。
mtob	n文字目までにあるバイト数を返す。
nthctype	nバイト目の文字の種類を返す。
zentohan	全角から半角へ変換する。

■グラフィック関数

GRAPH.H:

5	_arc	楕円の弧を描く。
5	_clearscreen	スクリーン領域の一部を消去する。
5	_displaycursor	カーソルの表示・非表示を設定する。

5	_ellipse	楕円を描く。
5	_floodfill	画面上の指定点から塗りつぶす。
5	_getbkcolor	バックグウンドカラーのピクセル値を返す。
5	_getcolor	カレントカラーを返す。
5	_getcurrentposition	現在のグラフィック出力位置の論理座標を返す。
5	_getfillmask	カレントフィルマスクを返す。
5	_getimage	指定領域のグラフィックイメージを格納する。
5	_getkanji	漢字フォントパターンを読み込む。
5	_getlinestyle	カレントラインスタイルマスクを返す。
5	_getlogcoord	物理座標を論理座標に変換する。
5	_getphyscoord	論理座標を物理座標に変換する。
5	_getpixel	指定した座標のピクセルを返す。
5	_gettextcolor	カレントテキストカラーを返す。
5	_gettextposition	現在のテキスト位置を返す。
5	_getvideoconfig	現在のグラフィック環境の状態を返す。
5	_gputchar	文字イメージをグラフィック画面に出力する。
5	_imagesize	スクリーンイメージを格納するのに必要なバイト数を返す。
5	_lineto	直線を引く。
5	_moveto	カレントポジションを設定する。
5	_outtext	文字列を出力する。(漢字)
5	_outtextg	文字列を出力する。(グラフィックキャラクタ)
5	_pie	扇形を描く。
5	_putimage	バッファの内容をスクリーンに出力する。
5	_rectangle	長方形を描く。
5	_remapallpalette	すべてのパレットを変更する。
5	_remappalette	ひとつのパレットを変更する。
5	_selectpalette	パレットを変更する。AXでのみ有効。
5	_setactivepage	アクティブページを選択する。
5	_setbkcolor	バックグラウンドカラーを設定する。
5	_setcliprgn	クリップ領域を設定する。
5	_setcolor	カラーを設定する。
5	_setfillmask	カレントフィルマスクを設定する。
5	_setkanji	外字パターンを登録する。
5	_setlinestyle	ラインスタイルを設定する。
5	_setlogorg	論理座標原点を任意の物理座標に移動する。
5	_setpixel	指定したピクセルをカレントカラーに設定する。
5	_settextcolor	テキストカラーを設定する。
5	_settextposition	テキスト座標位置を設定する。
5	_settextwindow	テキストウィンドウを指定する。
5	_setvideomode	スクリーンモードを選択する。
5	_setviewport	クリップ領域を設定する。
5	_setvisualpage	ビジュアルページを選択する。
5	_tilepaint	引数の配列で表わされた色で塗りつぶす。
5	_wrapon	ウィンドウボーダーでの処理を選択する。

リファレンス

The References

fopen

■書式

#include <stdio.h>

FILE ＊fopen (name, type)；

char ＊name；　ファイルネーム
char ＊type；　入出力モード

■戻り値

オープンしたストリームを指すファイルポインタを返します。
オープンに失敗した場合はNULLを返します。

■機能

fopen関数は、ファイル名nameのファイルをオープンします。
モードtypeはオープンのモードを決める文字列で、以下を使用します。

文字列	意味
"r"	読み出し専用です。ファイルが存在していなければエラーになります。
"w"	書き込み専用です。ファイルが存在する場合は削除します。
"a"	追加書き込み用です。ファイルが存在しなければ作成し、存在すればそのファイルの終端より追加します。
"r+"	読み書き両用です。ファイルが存在していなければエラーになります。
"w+"	読み書き両用です。ファイルが存在していた場合は削除されます。
"a+"	読み書き追加用です。ファイルが存在しなければ新たに作成します。

また、上記の文字列に以下の文字を付加することにより、復帰改行文字の取扱いを指定することができます。

文字	意味
t	テキストモードです。CR-LFと'¥n'の変換を行います。^Zはファイルの終端と見なされます。
b	バイナリモードです。文字の変換は行われません。

■ポイント

"r"や"w+"、"a+"などの「読み書き両用」のモードを指定すると、どちらの操作もできるようになりますが、読み書きする位置を変える時にはfsetpos, fseek, rewind関数などによって、現在のファイルポインタを変更する必要があります。

■例

¥autoexec.batに新しいディレクトリ検索パスを追加して、autoexec.newというファイルに書き出します。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *config = "¥¥autoexec.bat";
char *output = "autoexec.new";
char *path = "PATH";

main()
{
        FILE *fpi, *fpo;
        char buf[100], *pathenv;

        puts("autoexec.batの中に現在設定されているpath指定を");
        puts("付加します。出力はautoexec.newとなります。");
        pathenv = getenv(path);
        if (pathenv == NULL) {
                printf("PATHは定義されていません。¥n");
                exit(1);
        }
        if ((fpi = fopen(config, "r")) == NULL) {
                printf("%sがオープンできません。¥n", config);
                exit(1);
        }
        if ((fpo = fopen(output, "w")) == NULL) {
                printf("%sが作成できません。¥n", fpo);
                exit(1);
        }
        while (fgets(buf, 100, fpi) != NULL) {
                if (strstr(buf, "PATH") != NULL)
                        continue;
                if (strstr(buf, "path") != NULL)
                        continue;
                printf(buf);
                fputs(buf, fpo);
        }
        strcpy(buf, "path ");
        strcat(buf, pathenv);
        strcat(buf, "¥n");
        printf(buf);
        fputs(buf, fpo);
        fcloseall();
}
```

freopen

■書式

#include <stdio.h>

FILE ＊freopen (name, type, fp)；

char ＊name；　ファイルネーム
char ＊type；　入出力モード
FILE ＊fp；　ストリーム

■戻り値

新しくオープンしたストリームを指すファイル型のポインタを返します。
オープンに失敗した場合はNULLを返します。

■機能

freopen関数はストリームfpが指しているファイル等をクローズし、新しくファイルnameをfpと結び付け、オープンします。

通常、freopenは標準入出力ストリームのリダイレクトに使用します。

モードtypeは新たにオープンのモードを決める文字列で、以下を使用します。

文字列	意味
"r"	読み出し専用です。ファイルが存在していなければエラーになります。
"w"	書き込み専用です。ファイルが存在する場合は削除します。
"a"	追加書き込み用です。ファイルが存在しなければ作成し、存在すればそのファイルの終端より追加します。
"r+"	読み書き両用です。ファイルが存在していなければエラーになります。
"w+"	読み書き両用です。ファイルが存在していた場合は削除されます。
"a+"	読み書き追加用です。ファイルが存在しなければ新たに作成します。

また、上記の文字列に以下の文字を付加することにより、復帰改行文字の取扱いを指定することができます。

文字	意味
t	テキストモードです。CR-LFと'¥n'の変換を行います。^Zはファイルの終端と見なされます。
b	バイナリモードです。文字の変換は行われません。

■ポイント

標準入出力(stdin, stdout)に対しfreopenを行った場合は、子プロセスに継承されます。

エラーが起きた場合でも、ストリームfpはクローズされます。

■例

stdoutに対しfreopenを行い、puts関数の出力をファイルに書き込みます。

```
#include <stdio.h>

char *fil = "testfile";

main()
{
        /* stdoutをtestfileにリダイレクトする */
        FILE *fp;

        if ((fp = freopen(fil, "a", stdout)) == NULL) {
                fputs(fil, stderr);
                fputs("がオープンできません。¥n", stderr);
                exit(1);
        }
        puts("この文字列はstdoutに出力しています。");
        fclose(fp);
}
```

fclose
fcloseall

ver 5#

■書式

#include <stdio.h>

int fclose (fp);
int fcloseall (void);

FILE *fp;　　　ストリーム

■戻り値

ストリームのクローズに成功した場合、fclose関数は0を返し、fcloseall関数はクローズしたストリームの数を返します。

クローズに失敗した場合、どちらもEOFを返します。

■機能

fclose関数はストリームfpをクローズします。成功した場合は0を返し、失敗した場合はEOFを返します。

fcloseall関数はstdin, stdout, stderr, stdaux, stdprn以外のすべてのファイルをクローズします。成功した場合はクローズしたストリームの数を返し、失敗した場合はEOFを返します。また、fcloseall関数はtmpfile関数で作成した作業用ファイルも同時にクローズします。

■ポイント

fopen関数などでシステムが自動的に割り当てたバッファは、これらの関数により解放されますが、setvbufやsetbuf等の関数で割り当てられたバッファは解放されません。

■例

This is a sample.と書かれたファイルtest.tmpを作成します。

```
#include <stdio.h>

main()
{
        FILE *fp;
        static char fname[] = "test.tmp";

        if ((fp = fopen(fname, "w")) == NULL) {
                printf("%sが作成できませんでした。¥n", fname);
                exit(1);
        }
        fputs("This is sample.¥n", fp);
        if (fcloseall() == EOF) {
                printf("%sがクローズできませんでした。¥n", fname);
                exit(1);
        }
        printf("%sを作成しました。¥n", fname);
}
```

fdopen

■書式

```
#include <stdio.h>

FILE *fdopen (handle, type);

int handle;      ファイルハンドル
char *type;      入出力モード
```

■戻り値

オープンしたストリームを指すファイル型のポインタを返します。
オープンに失敗した場合はNULLを返します。

■機能

fdopen関数はファイルハンドルhandleに対してストリーム入出力が行えるように、ストリームとハンドルを結び付けます。

つまり、この関数によってファイルハンドルhandleに対して、ストリームを使用して入出力が可能になります。

モードtypeはオープンのモードを決める文字列で、以下を使用します。

文字列	意味
"r"	読み出し専用です。ファイルが存在していなければエラーになります。
"w"	書き込み専用です。ファイルが存在する場合は削除します。
"a"	追加書き込み用です。ファイルが存在しなければ作成し、存在すればそのファイルの終端より追加します。
"r+"	読み書き両用です。ファイルが存在していなければエラーになります。
"w+"	読み書き両用です。ファイルが存在していた場合は削除されます。
"a+"	読み書き追加用です。ファイルが存在しなければ新たに作成します。

また、上記の文字列に以下の文字を付加することにより，復帰改行文字の取扱いを指定することができます。

文字	意味
t	テキストモードです。CR-LFと'¥n'の変換を行います。^Zはファイルの終端と見なされます。
b	バイナリモードです。文字の変換は行われません。

■ポイント

通常、open関数等とともに使用します。

■例

コンソールをオープンし、文字列を出力します。

```
#include <stdio.h>
#include <fcntl.h>

int main(void)
{
        int handle;
        FILE *fp;

        handle = open("con", O_RDWR);
        if ((fp = fdopen(handle, "w+")) == NULL) {
                puts("conがオープンできません。");
                exit(1);
        }
        fputs("Microsoft (R) C Optimizing Compiler¥n", fp);
        fcloseall();

        return 0;
}
```

feof

■書式

#include <stdio.h>

int feof (fp);

FILE *fp;　　　ストリーム

■戻り値

ファイルの終端(EOF)に達している場合は0以外を返します。
それ以外の時は0を返します。

■機能

feofマクロはfpで指定されたファイルが終端に達しているかどうかを調べます。
EOFインジケータをリセットする場合には、rewindなどの関数を実行します。

■ポイント

feofマクロではエラー戻り値はありません。

■例

¥autoexec.batの内容を表示します。

```
#include <stdio.h>

char *fil = "¥¥autoexec.bat";

main()
{
        FILE *fp;
        int i;

        if ((fp = fopen(fil, "r")) == NULL) {
                perror(fil);
                exit(1);
        }
        while ((i = getc(fp), feof(fp)) == 0)
                putchar(i);
        fclose(fp);
}
```

ferror

■書式

```
#include <stdio.h>

int ferror (fp);

FILE *fp;        ストリーム
```

■戻り値

fpで指定されたストリームにエラーが発生しているときは0以外を返し、そうではないときは0を返します。

■機能

ferrorマクロはストリームfpにエラーが発生したかどうかを調べます。

エラーインジケータをリセットするには、ファイルをクローズするか、もしくはrewindやclearerr関数を実行します。

■ポイント

fgetcなどの関数では、エラーの発生でもファイルの終端に達した場合でも区別なくEOFを返します。そのような場合に、エラーであるかファイルの終端であるかを調べるためにこのマクロを使用します。

■例

¥autoexec.batの内容を表示します。エラーが発生したかどうかもチェックしています。

```
#include <stdio.h>

char *fil = "¥¥autoexec.bat";

main()
{
        FILE *fp;
        int i;

        if ((fp = fopen(fil, "r")) == NULL) {
                perror(fil);
                exit(1);
        }
        while ((i = getc(fp)) != EOF)
                putchar(i);
        if (ferror(fp))
                puts("ファイル読み込み中にエラーが発生しました。");
        fclose(fp);
}
```

fflush

■書式

#include <stdio.h>

int fflush (fp);

FILE *fp;　　　ストリーム

■戻り値

正常なときや、フラッシュする必要のないストリームの場合には0を返します。エラーが発生したときにはEOFを返します。

■機能

fflush関数はfpで指定されたストリームのバッファをフラッシュします。

フラッシュとはバッファの内容を掃除することで、出力用にオープンされたストリームであれば現在までのバッファの内容を書き込み、入力用にオープンされたストリームであればバッファの内容を消去します。

また、fflush関数はungetc関数で戻された文字も無効にします。

■ポイント

バッファは通常、以下の条件で自動的にフラッシュされます。

- バッファがいっぱいになったとき
- ストリームがクローズされたとき
- プログラムが正常終了したとき

fflush (stdin);

とすると現在までのキー入力バッファがクリアされます。

■例

2行を入力し、それぞれの先頭文字を表示します。

```
#include <stdio.h>

main()
{
        printf("char = %c.¥n", getchar());
        fflush(stdin);  /* 前回のgetcharの残りをクリアする */
        printf("char = %c.¥n", getchar());
}
```

flushall

■書式

＃include ＜stdio.h＞

int flushall (void)；

■戻り値

現在オープンしているストリームの総数を返します。返される値にはstdin, stdout等の標準入出力用のストリームも含まれます。エラー戻り値はありません。

■機能

flushall関数は現在オープンされているストリームのすべてにわたってフラッシュを行います。
出力用のストリームはバッファの内容を強制的にファイルへ書き込みます。
入力用のストリームはバッファの内容をクリアします。

■ポイント

この関数ではストリームがクローズされることはありません。
個々のファイルのフラッシュを行うときはfflush関数を使用します。

■例

子プロセスでオープンしているファイルを読み出し、フラッシュ前とフラッシュ後を比較しています。

```
#include    <stdio.h>
#include    <process.h>

void main(void)
{
        FILE    *fp;

        fp = fopen("test.tmp", "w");

        fputs("This is sample for flushall", fp);

        puts("type test.tmp (1)");
        system("type test.tmp");


        puts("type test.tmp (2)");
        flushall();
        system("type test.tmp");

        fclose(fp);
}
```

fscanf

ver 5#

■書式

#include <stdio.h>

int fscanf (fp, fmt, ...);

FILE *fp;　　　　ストリーム
const char *fmt;　　書式制御文字列

■戻り値

入力に成功したデータの個数(フィールドの数)を返します。
ファイルの終端(EOF)でfscanfが実行された場合、EOFを返します。
戻り値が0の場合は何も入力されなかったことを表わします。

■機能

fscanf関数は、書式制御文字列に従って、数値や文字列をストリームfpより入力します。
入力される変数は...の位置に記述しますが、与えるのは変数へのポインタです。
たとえば、10進数値を変数iに入力する場合は

```
fscanf (fp, "%d", &i);
```

のように引数を与えます。

■ポイント

書式制御文字列の指定はscanf関数と同様です。

■例

"2 3"と書かれたファイルを作成し、fscanf関数で数値として読み込みを行います。

```
#include <stdio.h>

char *fil = "tmpfile.tmp";

main()
{
        FILE *fp;
        int i, j;

        if ((fp = fopen(fil, "w")) == NULL) {
                printf("%sが作成できません。¥n", fil);
                exit(1);
        }
        fputs("2 3¥n", fp);
        fclose(fp);
        if ((fp = fopen(fil, "r")) == NULL) {
                printf("%sがオープンできません。¥n", fil);
                exit(1);
        }
        fscanf(fp, "%d %d", &i, &j);
        fclose(fp);
        printf("i = %d, j = %d¥n", i, j);
}
```

scanf

■書式

#include <stdio.h>

int scanf (fmt, ...)；

char ＊fmt；　　書式制御文字列

■戻り値

入力に成功したデータの個数(フィールドの数)を返します。ファイルの終端(EOF)でscanfが実行された場合はEOFを返します。戻り値が0の場合は何も入力されなかったことを表わします。

■機能

scanf関数は書式制御文字列に従って、数値や文字列をstdinより入力します。

入力される変数は**...**の位置に記述しますが、与えるのは変数へのポインタです。たとえば、10進数値を変数iに入力する場合は

scanf ("%d", &i)；

のように引数を与えます。

■ポイント

書式制御文字列の指定などは別表(printf関数の項)を参照してください。

■例

ふたつの数値を入力し、加減乗除を行います。

```
#include <stdio.h>

main()
{
        int i, j;

        printf("２つの整数をスペースで区切って入力してください¥n");
        scanf("%d %d", &i, &j);
        printf("%5d + %5d = %6d¥n", i, j, i + j);
        printf("%5d - %5d = %6d¥n", i, j, i - j);
        printf("%5d x %5d = %6d¥n", i, j, i * j);
        if (j == 0)
                printf("除算はできません。¥n");
        else
                printf("%5d / %5d = %6d¥n", i, j, i / j);
}
```

sscanf ver 5#

■書式

#include <stdio.h>

int sscanf (buf, fmt, ...)；

const char ＊buf；　文字列
const char ＊fmt；　書式制御文字列

■戻り値

代入に成功したデータの個数(フィールドの数)を返します。文字列の終端(EOF)でsscanfが実行された場合、EOFを返します。戻り値が0の場合は何も代入されなかったことを表わします。

■機能

sscanf関数は、書式制御文字列に従って、数値や文字列を文字列bufより代入します。

代入される変数は...の位置に記述しますが、与えるのは変数へのポインタです。たとえば、10進数値を変数iに代入する場合は

sscanf (str, "%d", &i)；

のように引数を与えます。

■ポイント

代入を文字列より行うことを除けばscanf関数と同じです。書式制御文字列については、別表(printf関数の項)を参照してください。

■例

文字列の内容を数値に変換し、簡単な演算の後、表示します。

```
#include <stdio.h>

main()
{
        int i, j;
        char *str = "24437 7";

        sscanf(str, "%d %d", &i, &j);
        printf("%d / %d = %d¥n", i, j, i / j);
}
```

fgetc
fgetchar

■書式

#include <stdio.h>

int fgetc (fp);
int fgetchar (void);

FILE *fp;　　　　ストリーム

■戻り値

いずれの関数も読み出した文字を返します。エラー発生時、もしくはファイルの終りではEOFを返します。

■機能

fgetc関数はfpで指定されたストリームから1文字読み込みます。
fgetchar関数はfgetc (stdin)と同じ動作をします。

■ポイント

fgetcやfgetchar関数の処理はgetcやgetcharと同じですが、fgetcやfgetcharは関数として定義されています。

■例

AからZまでの文字列を入力するタイプの練習プログラムです。

```
#include <stdio.h>

main()
{
        char *p;
        int c, er;
        static char str[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        printf("画面の通りにタイプしてリターンを押してください.¥n");
        do {
                er = 0;
                printf("¥033[34m%s¥033[m¥r", str);
                for (p = str; *p != '¥0'; p++) {
                        if ((c = fgetchar()) == EOF)
                                exit(1);
                        if (c < ' ')
                                continue;
                        if (c != *p) {
                                er++;
                                printf("¥033[36m%c¥033[m", c);
                        } else
                                putchar(c);
                }
                putchar('¥n');
        } while (er != '¥0');
        printf("¥nよくできました.¥n");
        exit(0);
}
```

fgets

■書式

```
#include <stdio.h>

char *fgets (buf, n, fp);

char *buf;      入力バッファ
int n;          バイト数
FILE *fp;       ストリーム
```

■戻り値

正常に読み込んだ場合はbuf(文字列の先頭を指すポインタ)を返します。
エラーもしくはファイルの終端の場合はNULLを返します。

■機能

fgets関数はストリームfpよりn文字までの1行を読み込みます。
fgets関数が1行と見なすのは、以下の条件のうち一番早いものによります。

- '¥n'が現れた
- ファイルの終端(EOF)に達した
- n－1バイトまで読み込んだ

読み込んだ文字列には'¥0'が付加されます。
バイト数に1を指定した場合は空文字列となります。

■ポイント

gets関数では'¥n'を'¥0'に置き換えますが、fgets関数ではそのまま保持し、文字列の最後に'¥0'を付加します。

■例

a:¥config.sysの内容を行番号付きで表示します。

```
#include <stdio.h>

char *fil = "a:¥¥config.sys";

main()
{
        FILE *fp;
        char buf[128];
        int i;

        i = 0;
        if ((fp = fopen(fil, "r")) == NULL) {
                printf("%sがオープンできません。", fil);
                exit(1);
        }
        while (fgets(buf, 128, fp) != NULL)
                printf("%3d:%s", ++i, buf);
        fclose(fp);
        exit(0);
}
```

fread

■書式

#include <stdio.h>

size_t fread (buf, size, count, fp);

void *buf;	データを格納するバッファ
size_t size;	1データあたりのバイト数
size_t count;	読み込む総データ数
FILE *fp;	ストリーム

■戻り値

正しく読み出せたデータの個数を返します。

エラーの場合は、エラーが発生する直前までに読み込んだ有効なデータの個数を返します。

■機能

fread関数はファイルから配列のデータを読み出す時などに使用します。

fread関数は1個のデータをsizeバイトとして、最大count個のデータをストリームfpから読み出し、bufが指す位置から書き込みます。

戻り値には、正しく読み出せたデータの個数を返します。

■ポイント

ストリームがテキストモードでオープンされている時は、fread関数は'¥n'の変換を行いますが、戻り値やファイルポインタがバイナリモードでオープンされている時と変わるようなことはありません。

■例

0から9までの自乗が格納された20バイトファイルの作成、読み込み、表示を行います。

```
#include <stdio.h>

char *tmp = "tmpfile.tmp";

main()
{
        FILE *fp;
        int buf[10], i;

        for (i = 0; i < 10; i++)
                buf[i] = i * i;
        if ((fp = fopen(tmp, "wb")) == NULL) {
                printf("ファイル%sが作成できません．¥n", tmp);
                exit(1);
        }
        fwrite(buf, sizeof(int), sizeof(buf), fp);
        fclose(fp);
        for (i = 0; i < 10; i++)
                buf[i] = 0;
        if ((fp = fopen(tmp, "rb")) == NULL) {
                printf("ファイル%sがオープンできません．¥n", tmp);
                exit(1);
        }
        fread(buf, sizeof(int), sizeof(buf), fp);
        fclose(fp);
        for (i = 0; i < 10; i++)
                printf("%1d^2 = %4d¥n", i, buf[i]);
}
```

getc
getchar

■書式
#include <stdio.h>

int getc (fp);
int getchar (void);

FILE　　*fp;　　ストリーム

■戻り値
読み出した文字を戻り値として返します。エラーやファイル終端の時は記号定数EOFを返します。

■機能
getc はストリームfpより1文字入力します。ファイルポインタはひとつ進みます。
getchar はgetc (stdin); と同じで、標準入力より1文字入力します。

■ポイント
getcとgetcharはマクロです。また、stdinは通常バッファリングされているので、

```
a = getchar ();
```

とだけした時でも、リターンキーが入力されるまで必ず待ってしまいます。

■例
大文字→小文字の変換を行うフィルタです。

```
#include        <stdio.h>

void main(void)
{
        int     c;

        while ((c = getchar()) != EOF) {
                if (c >= 'A' && c <= 'Z')
                        c += 32;
                putchar(c);
        }
}
```

gets

■書式

#include <stdio.h>

char ＊gets（buf）；

char ＊buf； 入力バッファ

■戻り値

引数bufの値をそのまま返します。エラーが発生した場合はNULLを返します。

■機能

gets関数はstdin（標準入力）から1行の文字列を読み込み、bufに格納します。
gets関数では'¥n'は'¥0'に変換されます。

■ポイント

NULLが返された場合は、ferrorやfeofによってエラーをチェックします。

■例

標準入力から文字列を受け取り、表示します。

```
#include <stdio.h>

main()
{
        char buf[80];

        printf("お名前をどうぞ!¥n");
        gets(buf);
        printf("%sさん、こんにちは。¥n", buf);
}
```

getw

■書式

#include <stdio.h>

int getw (fp);

FILE *fp;　　　ストリーム

■戻り値

読み出した整数値を返します。エラーが発生したか、ファイルの終端に達していた場合はEOFを返します。

■機能

getw関数はgetc関数のint版といった動作をする関数で、ストリームfpよりint型のデータを読み取り、戻り値として返します。

getc関数では1バイトのデータを読み出しますが、getw関数は1ワード(2バイト)のデータを読み出します。

■ポイント

記号定数EOFはgetw関数の正当な戻り値となり得る数値(つまり、int型の範囲内)が定義されていますので、実際のエラーかどうかはferrorなどによって調べなければなりません。

8086以外のCPU上で動く処理系との移植を行う場合、バイト並びの問題が発生する可能性がありますので注意が必要です。

■例

1から10までの数値の入った20バイトのファイルの作成、読み出し、表示を行います。

```
#include <stdio.h>

char *fil = "tmp.tmp";

void main(void)
{
        FILE *fp;
        int i;

        if ((fp = fopen(fil, "wb")) == NULL) {
                printf("%sが作成できません．¥n", fil);
                exit(1);
        }
        for (i = 0; i < 10; i++)
                putw(i, fp);
        fclose(fp);
        if ((fp = fopen(fil, "rb")) == NULL) {
                printf("%sが読み込めません．¥n", fil);
                exit(1);
        }
        for (i = 0; i < 10; i++)
                printf("%4d", getw(fp));
        fclose(fp);
        exit(0);
}
```

ungetc

■書式

#include <stdio.h>

int ungetc (c, fp);

int c;　　　　　文字
FILE *fp;　　　ストリーム

■戻り値

文字cを戻すことに成功した場合は文字cを返し、失敗した場合はEOFを返します。

■機能

ungetc関数はスリトームfpに文字cを戻し、EOFインジケータをクリアします。ストリームfpは読み出しが可能なモードでオープンされていなければなりません。

ungetc関数を使用して文字cをストリームに戻すと、直後のそのストリームに対する読み出しで文字cが読み出されます。ただし、読み出しの以前に以下の関数がそのストリームに対して行われた場合は消されてしまいます。

・fflush, fseek, fsetpos, rewind

■ポイント

テキストモードでオープンされているストリームでは、ungetc関数によって戻された文字を読み出している間、ファイルポインタは変化しません。

fprintf　　ver 5#

■書式

#include <stdio.h>

int fprintf (fp, fmt, ...)；

FILE ＊fp；　　ストリーム
const char ＊fmt；　　書式制御文字列

■戻り値

出力した文字の数を返します。

■機能

fprintf関数はストリームfpに対し、書式制御文字列に従って文字や数値を出力します。

■ポイント

出力先がストリームfpに指定できる以外はprintf関数と同様です。

■例

標準エラー出力(stderr)に文字列を出力します。

```
#include <stdio.h>

main()
{
        char *str = "この文字列は > ではリダイレクトできません。";

        fprintf(stderr, "str = %p, *str = '%s'¥n", str, str);
}
```

printf

■書式

```
#include <stdio.h>

int printf (fmt, ...);

FILE *fp;             ストリーム
const char *fmt;      書式制御文字列
```

■戻り値

出力した文字数を返します。

■機能

printf関数は書式制御文字列に従って、標準出力stdoutに直接文字や数値を出力します。

■ポイント

書式制御文字列については以下の別表を参照してください。

◉別表：書式指定文字列◉

★printf関数の書式指定文字列

% 【−】【+】【SPACE】【#】【width】【. prec】【model】【prefix】type

★scanf関数の書式指定文字列

% 【*】【width】【model】【prefix】type

記号	printf関数	scanf関数	意味
−	○		左揃え
+	○		数値に必ず符号(+/−)を付加する
スペース	○		数値が正ならばスペースをひとつ付加する
#	○		8/16進の数値の前に0,0x,0Xを付加する
*		○	このフィールドの分だけ代入を行わない
width	○	○	文字数の下限
prec	○		(全体から見た)有効桁数

記号	printf関数	scanf関数	文字	意味
model			文字	意味
	○	○	F	far型のポインタの指定
	○	○	N	near型のポインタの指定
prefix			文字	意味
	○	○	h	shortを意味する(typeを修飾する)
	○	○	l	longを意味する(typeを修飾する)
	○		L	long doubleを意味する(typeを修飾する)
type			文字	意味
	○	○	c	1文字
	○	○	d	符号付10進数値
	○		D	符号付10進long型数値
	○	○	e,E	符号付浮動小数点数値(指数形式)
	○	○	f	符号付浮動小数点数値
	○	○	g,G	e,fのコンパクトタイプ
	○	○	i	printf・・・符号付10進整数値 scanf・・・・符号付8,10,16進数値
		○	I	符号付8,10,16進数値
	○	○	n	整数を指すポインタ
	○	○	o	符号無し8進数値
		○	O	符号無し8進long型数値
	○	○	p	void型のfarポインタ
	○	○	s	文字列
	○	○	u	符号無し10進数値
		○	U	符号無し10進long型数値
	○	○	x,X	符号無し16進数値

注）E G X各typeでは、出力時にアルファベットが大文字になる。
　例：0xabcdを、typeにXを指定して出力 → ABCD

■例

簡単な文字列表示、数値表示を行います。

```
#include <stdio.h>

void main(void)
{
        printf("Hello, Microsoft (R) C Optimizing Compiler.¥n");
        printf(" 4 + 5 = %d¥n", 4 + 5);
        printf(" 2345 = 0x%x¥n", 2345);
        printf("%%4d & %%04d sample: %4d %04d¥n", 45, 45);
}
```

sprintf

ver 5#

■書式

#include <stdio.h>

int sprintf (buf, fmt, ...)；

char ＊buf；　文字列

const char ＊fmt；　書式制御文字列

■戻り値

bufに格納した文字数を返します。文字数に'¥0'は含まれません。

■機能

sprintf関数は書式制御文字列に従って、文字列bufに文字や数値を直接格納します。

■ポイント

変換された文字列がbufに格納される以外はprintf関数と同様です。書式制御文字列は別表(printf関数の項)を参照してください。

■例

簡単な演算結果を文字列に変換した後、出力を行います。

```
#include <stdio.h>

main()
{
        char buf[80];

        sprintf(buf, "56 * 22 = %d", 56 * 22);
        puts(buf);
}
```

fputc
fputchar

■書式

#include <stdio.h>

int fputc (c, fp);
int fputchar (c);

int c;　　　　文字
FILE *fp;　　ストリーム

■戻り値

どちらも、出力に成功した場合は文字cをそのまま返します。

失敗した場合はEOFを返しますが、文字cにEOFと同じ値を与えた場合もEOFを返しますので、その場合のエラーの確認はferror関数を使用します。

■機能

fputc関数はストリームfpに1文字cを書き込みます。

fputchar関数はfputc (c, stdout)として働きます。

■ポイント

fputcやfputchar関数の処理はputcやputcharと同じですが、putc,putcharがマクロであるのに対し、fputcやfputcharは関数として定義されています。

■例

¥autoexec.batの内容を表示します。

```
#include <stdio.h>

char *fil = "¥¥autoexec.bat";

main()
{
        FILE *fp;
        int c;

        if ((fp = fopen(fil, "r")) == NULL) {
                printf("%sがオープンできません。¥n", fil);
                exit(1);
        }
        while ((c = fgetc(fp), feof(fp)) == 0)
                fputchar(c);
        fclose(fp);
}
```

fputs

■書式

#include <stdio.h>

int fputs (str, fp);

const char *str;	文字列
FILE *fp;	ストリーム

■戻り値

文字列を全部正常に出力できた場合は0を返します。エラーの場合は0以外の値を返します。

■機能

fputs関数は文字列strをスリトームfpに出力します。ただし、文字列の終端マーク'¥0'は出力されません。

■ポイント

puts関数では文字列の最後に'¥n'が付加されて出力されますが、fputs関数では文字列には何も付加せずにそのまま出力します。

■例

"Hello, world."という文字列を2度出力します。改行の状態に注意してください。

```
#include <stdio.h>

main()
{
        char *str = "Hello, world.";

        puts(str);                      /* 改行される */
        putchar(':');
        putchar('¥n');

        fputs(str, stdout);             /* 改行されない */
        putchar(':');
        putchar('¥n');
}
```

fwrite

■書式

#include <stdio.h>

size_t fwrite (buf, size, count, fp)；

void ＊buf；	データが格納されているバッファ
size_t size；	1データ当りのバイト数
size_t count；	書き込む総データ数
FILE ＊fp；	ストリーム

■戻り値

正しく書き込めたデータの個数を返します。

エラーの場合は、エラーが発生する直前までに書き込んだ有効なデータの個数を返します。

■機能

fwrite関数はファイルに配列のデータを書き込むときなどに使用します。

fwrite関数は1個のデータをsizeバイトとして、bufが指す位置からの最大count個のデータをストリームfpへ書き込みます。

戻り値には正しく書き込めたデータの個数を返します。

■ポイント

ストリームがテキストモードでオープンされている時は、fwrite関数は'¥n'の変換を行いますが、戻り値やファイルポインタがバイナリモードでオープンされている時と変わるようなことはありません。

■例

fread関数を参照してください。

putc
putchar

■書式

#include <stdio.h>

int putc (c, fp);
int putchar (c);

int c;	文字
FILE *fp;	ストリーム

■戻り値

どちらも出力に成功した場合は、文字cをそのまま返します。

失敗した場合はEOFを返しますが、文字cにEOFと同じ値を与えた場合もEOFを返しますので、実際のエラーの確認はferror関数を使用します。

■機能

putc はストリームfpに1文字cを書き込みます。

putchar はputc (c, stdout)として働きます。

■ポイント

putc,putcharとfputc,fputcharは同じ働きをしますが、putc,putcharはマクロです。

■例

標準出力および標準エラー出力に1文字ずつ文字列を表示します。

```
#include        <stdio.h>
main()
{
        char    *ptr1 = "この文字列はリダイレクトできます。¥n";
        char    *ptr2 = "この文字列はリダイレクトできません。¥n";

        while(*ptr1 != '¥0') {
                putchar(*ptr1);
                ptr1++;
        }

        while(*ptr2 != '¥0') {
                putc(*ptr2, stderr);
                ptr2++;
        }
}
```

puts

■書式

#include <stdio.h>

int puts (str);

const char *str;　　　文字列

■戻り値

文字列を全部正常に出力できた場合は0を返します。エラーの場合は0以外の値を返します。

■機能

puts関数は文字列strを標準出力stdoutに出力します。ただし、文字列の終端マーク'¥0'は出力されません。また、文字列の終端に'¥n'が付加され、自動的に改行します。

■ポイント

puts関数では文字列の最後に'¥n'が付加されて改行されます。改行させたくない場合は、fputs(str, stdout);のようにfputs関数を使用します。

puts関数はprintf関数と比較してライブラリが小さいため、printf関数で数値を表示しない場合はputs関数に置き換えることによって、サイズを縮めることができます。

■例

fputs,puts関数を用いて標準出力に文字列を表示します。

```
#include <stdio.h>

main()
{
        puts("これは、puts関数の出力です。");
        fputs("これは、fputs関数の出力です。", stdout);
        puts("←fputs関数では改行しません。");
}
```

putw

■書式
```
#include <stdio.h>

int putw (i, fp);

int i;          書き込む整数値
FILE *fp;       ストリーム
```

■戻り値

書き込んだ整数値を返します。エラーが発生したか、ファイルの終端に達していた場合はEOFを返します。

■機能

putw関数はputcマクロのint版といった働きをします。putw関数は、ストリームfpにint型のデータiを出力します。(putcでは、char型のデータを出力します)

putw関数で書き込むストリームは、通常バイナリモードでオープンします。なぜならば、putw関数に与えられたiの値によっては'¥n'や'¥0'などの特殊な文字を含むこともあるからです。

■ポイント

記号定数EOFはputw関数の正当な戻り値となり得る数値(つまり、int型の範囲内)が定義されていますので、実際のエラーかどうかはferrorなどによって調べます。

8086以外のCPU上で動く処理系との移植を行う場合、バイト並びの問題が発生する可能性がありますので注意が必要です。

■例

getw関数を参照してください。

fseek

■書式
#include <stdio.h>

int fseek (fp, ofs, pos);

FILE *fp;	ストリーム
long ofs;	相対位置
int pos;	参照位置

■戻り値
ファイルポインタの移動に成功したときは0を返します。エラーが発生したときは0以外の値を返します。
デバイスなどのファイルポインタが移動できないストリームに対してfseekを行った場合、戻り値は不定となります。

■機能
fseek関数はストリームfpの現在のファイルポインタ(ストリームの中の現在の入出力位置を指す内部ポインタ)を、posより相対的にofsバイトの位置に移動します。
ofsには符号付の整数でposで参照する位置からの相対的なバイト数を指定します。
posには以下の記号定数を指定します。

記号定数	意味
SEEK_SET	ファイルの先頭
SEEK_CUR	ファイルポインタの現在位置
SEEK_END	ファイルの終端

■ポイント
fseek関数はEOFインジケータをクリアします。また、fseek関数より以前にストリームfpに対して行われたungetcはすべて無効になります。
テキストモードでオープンされたストリームに対してfseek関数を使用すると、fseek関数での位置計算が複雑になるため、そのようなストリームにfseek関数を使用するのは以下のような場合のみにします。

- 相対位置ofsに0を指定する場合
- ftell関数によって得た値をofsとし、posにSEEK_SETを指定する場合

■例

config.sys内の最後のdevice指定を表示します。

```
#include <stdio.h>
#include <string.h>

char *fil = "¥¥config.sys";

main()
{
        FILE *fp;
        long tmp, posit;
        char buf[120];

        posit = -1L;
        if ((fp = fopen(fil, "rb")) == NULL) {
                printf("%sがオープンできません。¥n", fil);
                exit(1);
        }
        while ((tmp = ftell(fp), fgets(buf, 120, fp)) != NULL) {
                if (strstr(buf, "device") != NULL ||
                                    strstr(buf, "DEVICE") != NULL)
                        posit = tmp;
                printf("%s", buf);
        }
        if (posit == -1L)
                printf("¥ndevice指定がありません。¥n");
        else {
                fseek(fp, posit, SEEK_SET);
                fgets(buf, 120, fp);
                fclose(fp);
                printf("最後のdevice指定は¥n%sです。¥n", buf);
        }
        fclose(fp);
        exit(0);
}
```

ftell

■書式

```
#include <stdio.h>

long ftell (fp);

FILE *fp;          ストリーム
```

■戻り値

現在のファイルポインタの位置をlong型で返します。
エラーが起こった場合は－1Lを返し、errnoに以下の記号定数を返します。

記号定数	エラー内容
EBADF	ストリームfpの指定が不正です。
EINVAL	ストリームfpフハンドル値が無効です。

■機能

ftell関数はストリームfpの現在のファイルポインタの位置を返します。

■ポイント

ファイルポインタを移動できないようなストリームを指定すると、戻り値は不定になります。
テキストモードのストリームを指定したときは、通常fseek関数等とともに使用します。

■例

fseekを参照してください。

fgetpos　ver 5

■書式

```
#include <stdio.h>

int fgetpos (fp, pos);

FILE *fp;       ストリーム
fpos_t *pos;    ファイル位置格納域
```

■戻り値

ファイルポインタの位置をposに取得できた時は0を返します。

エラーが発生した場合は0以外の値を返し、errnoに下記の記号定数を返します。

記号定数	エラー内容
EINVAL	ストリームfpが不正である。
EBADF	ストリームfpのハンドル値が不正か、ファイルにアクセスできない。

■機能

fgetpos関数はストリームfpの現在のファイルポインタ(ファイル先頭からの位置)の位置を取得し、fpos_t型のポインタposの指す位置に返します。

■ポイント

posは変数ではなく変数のアドレス(または変数を指すポインタ)を与えることに注意してください。たとえば、

```
fpos_t pos;
fgetpos (fp, pos);
```

は誤りであり、正しくは

```
fpos_t pos;
fgetpos (fp, &pos);
```

と記述します。

fsetpos　　ver 5

■書式

```
#include <stdio.h>

int fsetpos (fp, pos);

FILE *fp;              ストリーム
const fpos_t *pos;     ファイル
```

■戻り値

成功した場合は0を返します。
エラーが発生した場合は0以外の値を返し、errnoに下記の記号定数を返します。

記号定数	エラー内容
EINVAL	ストリームfpが不正である。
EBADF	ストリームfpのハンドル値が不正か、ファイルにアクセスできない。

■機能

fsetpos関数はストリームfpのファイルポインタをposで指定された位置にセットします。
fsetpos関数は同時にEOFインジケータをクリアし、ストリームfpに行われたungetc関数を無効にします。

■ポイント

posには事前にfgetpos関数で与えられた数値を使用してください。

rewind

■書式

#include <stdio.h>

void rewind (fp)；

FILE ＊fp；　　　ストリーム

■戻り値

ありません。

■機能

rewind関数は、ストリームfpのファイルポインタをファイルの先頭に戻します。
また、rewind関数はEOFインジケータやエラーインジケータをクリアします。

■ポイント

fseek (fp, 0L, SEEK_SET)；と同じ働きをしますが、fseekではエラーインジケータをクリアしません。

■例

¥autoexec.batの内容を2度繰り返し表示します。

```
#include <stdio.h>

main()
{
        FILE *fp;
        int c;
        char *fname = "¥¥autoexec.bat";

        if ((fp = fopen(fname, "r")) == NULL) {
                perror(fname);
                exit(1);
        }
        while ((c = fgetc(fp)) != EOF)
                putchar(c);
        rewind(fp);
        while ((c = fgetc(fp)) != EOF)
                putchar(c);
        fclose(fp);
}
```

clearerr

■書式

```
#include <stdio.h>

void clearerr (fp);

FILE *fp;        ストリーム
```

■戻り値

ありません。

■機能

clearerr関数はストリームfpのEOFやエラーインジケータをリセットします。

■ポイント

EOFやエラーインジケータは、一度値がセットされる(エラーが起きる)と自動的にクリアされることはありません。強制的にクリアしたい時にこの関数を使用します。

fileno

■書式

#include <stdio.h>

int fileno (fp)；

FILE ＊fp；　　　ストリーム

■戻り値

ストリームfpと結合しているファイルハンドルを返します。

■機能

filenoマクロはストリームfpと結合しているファイルハンドルを返します。
fpに無効なストリームを指定した場合、戻り値は保証されません。

■ポイント

ストリームが複数のハンドルと結合している場合には、最初に結合されたハンドルを返します。

■例

何も行わないプログラムです。EOFコードが途中に含まれても、標準入力から標準出力に全内容が出力されます。

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>

void main(void)
{
        int c;

        setmode(fileno(stdin), O_BINARY);
        setmode(fileno(stdout), O_BINARY);
        while ((c = getchar()) != EOF)
                putchar(c);
}
```

rmtmp　ver 4

■書式

#include ＜stdio.h＞

int rmtmp (void)；

■戻り値

削除されたテンポラリファイルの個数を返します。

■機能

rmtmp関数はカレントディレクトリにあるtmpfile関数で作成したすべてのテンポラリファイルをクローズおよび削除します。

■ポイント

rmtmp関数を使用する場合、必ずtmpfile関数で作成したファイルが存在するディレクトリに移動していなければなりません。

setbuf

■書式

```
#include <stdio.h>

void setbuf (fp, buf);

FILE *fp;        ストリーム
char *buf;       バッファ
```

■戻り値

ありません。

■機能

setbuf関数はストリームfpのバッファを変更します。
setbuf関数により、以後ストリームfpの入出力用バッファはbufになります。
ユーザーはbufにBUFSIZバイトだけの領域を確保しなければなりません。
bufにNULLを指定することで、ストリームfpをバッファリングしないようにセットすることができます。

■ポイント

stdauxとstderrはデフォルトではバッファリングしませんが、setbuf関数によりバッファリングも可能になります。

setvbuf　　ver 4, ver 5#

■書式

```
#include <stdio.h>

int setvbuf (fp, buf, type, size);

FILE *fp;        ストリーム
char *buf;       バッファ
int type;        バッファの型
size_t size;     バッファのサイズ
```

■戻り値

正常にバッファを設定できた場合は0を返します。型が無効であったり、バッファが無効であった等のエラーの場合は0以外を返します。

■機能

setvbuf関数はストリームfpの入出力用バッファとしてbufを使用し、typeで指定された方法によってバッファリングを行います。sizeにはバッファbufのサイズをバイト単位で指定します。

bufにNULLが指定された場合、setvbuf関数は自動的にsizeバイトのメモリをバッファのために確保します。

typeには以下の記号定数を指定します。

記号定数	意味
_IOFBUF	フルバッファリングします。
_IOLBUF	_IOFBUFと同じです。
_IONBF	バッファリングしません。bufとsize引数は無視されます。

■ポイント

MS-C ver4.0では、bufがNULLの場合はバッファリングを行いませんでした。

tempnam
tmpnam

ver 4

■書式

```
#include <stdio.h>

char *tempnam (tmpdir, name);
char *tmpnam (name);

char *tmpdir;    ディレクトリ
char *name;      ファイルネーム
```

■戻り値

作成されたファイルネームを指すポインタを返します。ファイルネームが作成できなかったか、すでに同名のファイルが存在していた場合はNULLを返します。

■機能

tmpnam関数は一時ファイルとして使用できるテンポラリファイル名を生成します。生成された名前はnameに格納されます。nameにNULLが指定された場合、内部の静的バッファに名前が残され、その位置を指すポインタが返されます。

nameには、少なくともL_tmpnamバイトの領域を確保しなくてはなりません。

tempnam関数はテンポラリファイルのディレクトリを指定する場合に使用します。nameはファイル名の先頭の文字列となります。

まずtempnam関数は作成したファイル名が以下のディリクトリに存在しないか、チェックします。

- 環境変数TMPが指定されていて、指定されているディレクトリが存在する場合は、そのディレクトリ
- 引数tmpdirがNULLでなく、かつ、そのディレクトリが存在する場合は、そのディレクトリ
- P_tmpdir (stdio.hで定義されています)が存在する場合は、そのディレクトリ
- 上記の条件にあてはまらない場合は、カレントディレクトリ

上記のディレクトリとテンポラリファイル名を結合したものをmalloc関数によって、自動的にメモリ上に生成します。ユーザーは、このファイル名が必要でなくなったときは、free関数などによって解放しなくてはなりません。

■ポイント

tmpnam関数で生成される文字列はP_tmpdir(stdio.hで定義されています)のパス名と、これに続く一連のint型整数値です。

tmpfile　　ver 4, ver 5#

■書式

#include <stdio.h>

FILE *tmpfile (void);

■戻り値

テンポラリファイルの作成に成功した場合はそのストリームへのポインタを返し、失敗した場合はNULLを返します。

■機能

tmpfile関数はテンポラリファイルを作成します。

テンポラリファイルは"w+b"モードでオープンされます。

■ポイント

この関数を使用してテンポラリファイルを使用した後、rmtmp関数などによってテンポラリファイルを破棄する場合は、カレントディレクトリをこの関数が実行された時点でのカレントディレクトリに移動しておかなければなりません。

■例

¥autoexec.batの内容をテンポラリファイルと標準出力に出力します。

```
#include <stdio.h>
#include <ctype.h>

char *nam = "¥¥autoexec.bat";

main()
{
        FILE *fpi, *fpo;
        int i;

        if ((fpi = fopen(nam, "r")) == NULL) {
                perror(nam);
                exit(1);
        }
        if ((fpo = tmpfile()) == NULL) {
                puts("テンポラリファイルが作成できません。");
                fcloseall();
                exit(1);
        }
        while ((i = fgetc(fpi)) != EOF)
                fputc(tolower(i), fpo);
        fclose(fpi);
        rewind(fpo);
        while ((i = fgetc(fpo)) != EOF)
                putchar(i);
        rmtmp();
}
```

vprintf
vfprintf
vsprintf

ver 4

■書式

#include <stdio.h>
#include <varargs.h>
#include <stdarg.h>

int vprintf (format, argptr);
int vfprintf (stream, format, argptr);
int vsprintf (buf, format, argptr);

char *format;	書式制御文字列
va_arg argptr;	別数並びを指すポインタ
FILE *stream;	ストリーム
char *buf;	出力するバッファ

■戻り値

書き込まれた文字数を返します。vfprintfの場合、エラーが発生すると－1を返します。

■機能

可変引数による文字列の書式付き出力を行います。引数が可変引数であることを除き、vprintf, vfprintf, vsprintfは、printf, fprintf, sprintfに相当する機能を持ちます。

■ポイント

UNIX System Vとコンパチブルの可変引数マクロを用いる場合はvarargs.h、ANSI規格案とコンパチブルな可変引数マクロを用いる場合はstdarg.hを使用します。双方を同時に使用することはできません。

■例

以下の例ではvfprintfを用いて、標準エラー出力にフォーマット出力を行う関数eprintfを定義しています。UNIX System Vコンパチブルのvarargs.hを使用しています。

```
#include <stdio.h>
#include <varargs.h>

int eprintf(va_alist)
va_dcl
{
        char *fmt;
        va_list argp;

        va_start(argp);
        fmt = va_arg(argp, char *);
        vfprintf(stderr, fmt, argp);
        va_end(argp);
}


void main(int argc, char **argv)
{
        eprintf("%s(%d) %s¥n", argv[0], argc, "stderr / this is errormessage");
}
```

kbhit

■書式

#include ＜conio.h＞

int kbhit（void）；

■戻り値

キー入力が行われていれば戻り値として0以外の値を返します。キー入力がなければ0を返します。

■機能

kbhit関数はコンソール(バッファ)に文字が入力されているかどうかを調べる関数です。キーが押されていたら(つまり、バッファに文字がたまっていたら)0以外の値を返し、押されていなかったら(つまり、バッファに文字がたまっていなかったら)0を返します。

■ポイント

リアルタイムでキー入力を判定するときなどに便利です。

■例

画面上を'＊'が移動します。何かキーを押すと終了します。

```
#include        <conio.h>
#include        <stdio.h>

main()
{
        int     x, y, a, b;

        printf("¥033[2J何かキーを押せばとまります.");

        x = y = 2;
        a = b = 1;
        printf("¥033[%d;%dH*", y, x);


        while(kbhit() == 0) {
                printf("¥033[%d;%dH ", y, x);
                x += a;
                y += b;

                if (x < 2 || x > 77)
                        a = -a;
                if (y < 2 || y > 20)
                        b = -b;
                printf("¥033[%d;%dH*", y, x);
        }
}
```

getch
getche

■書式

#include ＜conio.h＞

int getch (void)；
int getche (void)；

■戻り値

それぞれの関数は入力された文字を返します。エラー戻り値はありません。

■機能

getch, getche関数は直接コンソールから文字を1文字入力します。getche関数は入力された文字をその場でエコーバックしますが、getch関数ではエコーバックはありません。

■ポイント

どちらの関数も＾Cを読み出すことはできません。

■例

パスワード付きでプログラムを起動します

```
#include        <stdio.h>
#include        <conio.h>
#include        <string.h>
#include        <process.h>

void main(int argc, char **argv)
{
        char    *passwd = "MS-C";
        char    nam[50];

        if(argc == 1) {
                cprintf("実行したいプログラムの名前をオプションとして"
                "指定して下さい。¥r¥n");
                exit(0);
        }

        cprintf("Your name:");
        nam[0] = 47;
        cgets(nam);

        cprintf("¥r¥nPassword :");
        while(*passwd != '¥0') {
                if(getch() != *passwd) {
                        cprintf("¥r¥nYou are not allowed to execute.¥r¥n");
                        exit(1);
                }
                passwd++;
        }

        cprintf("¥r¥nHello,%s!¥r¥n", &nam[2]);
        system(argv[1]);
}
```

cgets

■書式

```
#include <conio.h>

char *cgets (str);

char *str;        文字列バッファ
```

■戻り値

(str ＋ 2)、str[2]のアドレスを返します。str[2]からは、実際に入力された文字がセットされています。

■機能

cgets関数はコンソールから1行読み込みます。

cgets関数を使用する場合は、あらかじめstr[0]に読み込む文字列の最大長をセットしておきます。これは、'¥0'を含めない長さなので、実際には更に1文字分確保する必要があります。

cgets関数は文字を1行分読み込むと、入力されたCR/LFを'¥0'に変換し、str[1]に入力された文字の長さがセットされます。

■ポイント

cgets関数の戻り値はstrでないことに気を付けてください。&str[2]、つまり文字列の先頭を指すポインタを返します。

■例

コンソールから文字列を入力し、その長さと文字列を表示します。

```
#include        <conio.h>
#include        <stdio.h>

main()
{
        char    buf[80];
        buf[0] = 77;            /* 最大長さ */

        cgets(buf);

        printf("文字列の長さ=%d¥n", buf[1]);
        printf("文字列:¥n%s¥n", &buf[2]);
}
```

cscanf　　ver 5#

■書式

#include <conio.h>

int cscanf (fmt, ...);

char　*fmt;　書式制御文字列

■戻り値

入力されたデータ(フィールド)の個数を返します。ファイルの終端の場合は、EOFを返します。

■機能

cscanf関数は書式制御文字列に従って、コンソールよりデータを入力します。

■ポイント

cscanf関数はコンソールからダイレクトに入力を行う他は、scanfと同等の機能を持っています。

putch　　ver 5#

■書式

#include <conio.h>

int putch (c);

int c;　　　　文字

■戻り値

表示できた場合は文字cをそのまま返し、表示できなかった場合はEOFを返します。

■機能

putch関数はコンソールに直接、文字cを書き込みます。そのため、'¥n'は復改コード(CR・LF)に直されません。

■ポイント

MS-Cバージョン4.0では、putchはエラーコードを返します。

■例

色を変えて文字列を出力します。

```
#include        <conio.h>
#include        <stdio.h>

main()
{
        char    *ptr = "¥033[36m This is sample.¥033[m¥r¥n";

        while(*ptr != '¥0')
                putch(*(ptr++));
}
```

ungetch

■書式

＃include ＜conio.h＞

int ungetch (c)；

int c；　　　　文字

■戻り値

文字cを戻すことに成功した場合は文字cを返し、失敗した場合はEOFを返します。

■機能

ungetch関数はgetch等の関数で、あたかも文字cを入力したかのように「文字を戻す」関数です。
文字cはEOFであってはいけません。

■ポイント

ungetch関数は文字入力をせずに2回以上使用してはいけません。
また、getchar()関数等には無効です。

■例

'S'が入力されたかのように見せかけます。

```
#include        <conio.h>
#include        <stdio.h>

main()
{
        ungetch('S');
        printf("文字 %c が入力されました。¥n", getch());
}
```

cputs　ver 5#

■書式

#include <conio.h>

int cputs (str) ;

char *str ;　　文字列

■戻り値

表示できたときは0を返します。表示できなかった時は0以外の値を返します。

■機能

cputs関数はコンソールに文字列strを表示します。

■ポイント

cputs関数はputs関数と違い、'¥n'を最後に付加して表示しません。改行させたい時は'¥r ¥n'を付加してください。

■例

画面に文字列を表示します。

```
#include        <conio.h>
#include        <stdio.h>

main()
{
        char    *str1 = "文字列１";
        char    *str2 = "文字列２¥r¥n";

        cputs(str1);
        cputs(str2);
        cputs(str1);
        cputs("¥r¥n");
}
```

cprintf　　ver 5#

■書式

#include <conio.h>

int cprintf（fmt, ...）;

char　*fmt;　書式制御文字列

■戻り値

表示した文字の数を返します。

■機能

cprintf関数は書式制御文字列に従って、コンソールに直接文字や数値を出力します。

cprintf関数では、'¥n'を出力する際に復改コード(CR・LF)には変換しません。したがって、復帰改行する時には"¥r¥n"とします。

■ポイント

cprintf関数は文字の出力の際にputch関数を使用している以外、printf関数と同等の機能を持っています。

inp
inpw

[inpw]ver 5

■書式

```
#include <conio.h>

int inp (port);
unsigned inpw (port);

unsigned int port;          ポート番号
```

■戻り値

ポートportから読み出したデータを返します。inpはバイトデータを、inpwは符号なしワードデータを返します。

■機能

inpおよびinpw関数はCPUのI/Oポートのポート番号portよりデータを読み込みます。

■ポイント

ポート番号は0から65535までを指定します。

outp
outpw

(outpw)ver 5

■書式

```
#include <conio.h>

int outp (port, b_data);
unsigned outpw (port, w_data);

unsigned int port;     ポート番号
int b_data;            バイトデータ
unsigned int w_data;   ワードデータ
```

■戻り値

出力したデータをそのまま返します。

■機能

outpおよびoutpw関数はCPUのI/Oポートのポート番号portへデータを出力します。

■ポイント

ポート番号は0から65535までを指定します。

open

■書式

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

int open (path, oflag, [, pmode]);

char *path;     ファイルのパス名
int oflag;      オープンモード
int pmode;      アクセス許可
```

■戻り値

オープンされたファイルのファイルハンドルを返します。−1はエラーを示し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	指定パス名はディレクトリです。または、指定ファイルが読み出し専用なのに、書き込みモードでオープンしようとした。または、シェアリングモードが不正。
EEXIST	O_CREAT, O_EXCLを指定した時、同名のファイルが既に存在しています。
EMFILE	オープンしているファイルが多すぎます。
ENOENT	ファイル名またはパス名が見つかりません。

■機能

pathで指定したファイルをオープンします。

oflagにはアクセスモードを指定します。これはfcntl.hに次のように定義された定数を組み合わせて指定します。

記号定数	意味
O_APPEND	ファイルポインタをファイルの最後に移動します。
O_BINARY	バイナリモードでオープンします。
O_CREAT	新たにファイルを作成します。
O_EXCL	ファイルが既に存在している場合はエラーを返します。
O_RDONLY	読み出し専用でオープンします。
O_RDWR	読み書き可能でオープンします。
O_TEXT	テキストモードでオープンします。

O_TRUNC　既に存在しているファイルの内容を破棄します。
O_WRONLY　書き込み専用でオープンします。

O_RDONLY, O_RDWR, O_WRONLYのどれかひとつは必ず指定してください。

pmodeはO_CREATを指定してファイルの作成を行う場合に指定します。
これらの定数はsys/stat.hに定義されています。

記号定数	意味
S_IWRITE	書き込み可能
S_IREAD	読み出し可能
S_IREAD ｜ S_IWRITE	読み書き可能

■**ポイント**

MS-DOSでは読み出し禁止の属性はありませんから、pmodeのS_IREADは意味を持ちません。
アクセス許可を設定する前に、pmodeに対する現在のファイル許可設定を適用します。これはumask関数で設定できます。

■**例**

read関数を参照してください。

sopen

■書式

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <share.h>
#include <io.h>

int sopen (path, oflag, shflag, [, pmode]);

char *path;      ファイルのパス名
int oflag;       アクセスモード
int shflag;      ファイルシェアリングの種類
int pmode;       アクセス許可
```

■戻り値

オープンしたファイルのファイルハンドルを返します。エラーの場合は−1を返し、errnoにopen関数と同様の値を設定します。

■機能

pathで指定したファイルをオープンし、oflagおよびshflagで指定したモードでファイルを共有した読み書きに備えます。OS/2またはMS-DOSのバージョン3.xのネットワーク環境下で、指定されたシェアリングモードでファイルをオープンするのに使用します。

oflag およびpmodeはopen関数と同様です。

shflagには、share.hで定義されている次の定数のいずれかを指定します。

記号定数	意味
SH_COMPAT	互換モードにします。
SH_DENYRW	読み書きを拒否します。
SH_DENYWR	書き込みを拒否します。
SH_DENYRD	読み出しを拒否します。
SH_DENYNO	読み書きを許可します。

■ポイント

SHAREコマンドがインストールされていない場合は、MS-DOSはシェアリングモードを無視します。

■例

指定されたファイルを、MS-DOSのバージョンが3.0以降の場合についてのみ、シェアリングモードでオープンします。

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <share.h>
#include <io.h>

extern unsigned char _osmajor;

int openfile(path)
char *path;
{
        if (_osmajor >= 3)
                return sopen(path, O_RDWR, SH_DENYRW);
        else
                return open(path, O_RDWR);
}
```

close

■書式

#include <io.h>

int close (handle);

int handle;　　ファイルハンドル

■戻り値

正常なら0、エラーが起ったら−1を返し、errnoに次の値を返します。

記号定数	エラー内容
EBADF	指定したファイルハンドルが間違っています。

■機能

handleで指定したファイルをクローズします。

■ポイント

通常open関数でオープンしたファイルをクローズするのに使います。

■例

read関数を参照してください。

creat

■書式

```
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

int creat (path, pmode);

char *path;      ファイルへのパス名
int pmode;       アクセス許可の設定
```

■戻り値

正常ならファイルハンドルを返します。エラーが起きた場合は−1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	指定ファイルは書き込み禁止、またはディレクトリです。
EMFILE	ファイルハンドルが割り当てられません。オープン中のファイルが多すぎます。
ENOENT	パス名が見つかりません。

■機能

pathで指定したファイルを作成します。その時のファイル属性はpmodeに従って設定されます。pathで指定したファイルが既に存在し、書き込み許可であった場合は、ファイルサイズを0に設定して、新たな書き込み操作のためにオープンします。

pmodeに指定する定数はsys/stat.hに次のように定義されています。

記号定数	意味
S_IWRITE	書き込み可能
S_IREAD	読み出し可能
S_IREAD | S_IWRITE	読み書き可能S_IWRITE

これらの意味はopen関数と同様です。

■ポイント

MS-DOSでは読み出し禁止の属性はありませんから、pmodeのS_IREADは意味を持ちません。

■例

与えられたファイル名だけファイルを作成するコマンドです。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
#include <stdio.h>

void main(ac, av)
int ac;
char **av;
{
        while (--ac) {
                if (creat(*++av, S_IREAD | S_IWRITE) == -1) {
                        fprintf(stderr, "cannot creat '%s'¥n", *av);
                        exit(1);
                }
        }
        exit(0);
}
```

write

■書式

#include <**io.h**>

int write (handle, buffer, count)；

int handle；	ファイルハンドル
char ＊buffer；	書き込むデータ
unsigned int count；	バイト数

■戻り値

書き込んだバイト数を返します。−1はエラーを示し、 errnoに次の値を返します。

記号定数	エラー内容
EBADF	ファイルハンドルが無効。
ENOSPC	デバイスに空き領域が無い。

■機能

handleに対応するファイルの現在のファイルポインタの位置から、bufferの指す領域をcountバイト数書き込みます。書き込み処理後のファイルポインタは、実際に書かれたバイト数分だけ進められます。

■ポイント

テキストモードでオープンしたファイルに対しては改行文字'¥n'は復帰改行'¥r¥n'に置き換えられます。ただし、戻り値には含まれません。

32Kbytes以上の書き込みを行う場合は、戻り値をunsigned intとしてください。また、戻り値65535は−1と区別ができないので、書き込める最大バイト数は65534(0xFFFE)となります。

■例

5バイト書き込んだのに対し、改行が展開されてファイルポインタは6バイト移動しているのがわかります。

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>

void main()
{
        int fd;
        static char data[] = "test¥n";

        fd = open("temp", O_RDWR | O_CREAT | O_TRUNC,
                                        S_IREAD | S_IWRITE);
        write(fd, data, 5);
        printf("pos = %ld¥n", tell(fd));
        close(fd);
}
```

read

■書式

```
#include <io.h>

int read (handle, buffer, count);

int handle;              ファイルハンドル
char *buffer;            データ格納領域
unsigned int count;      読み込むバイト数
```

■戻り値

読み出したバイト数を返します。エラーの場合は−1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EBADF	指定したhandleが無効、またはファイルが読み出し不許可です。

■機能

handleで指定したファイルの現在のファイルポインタからbufferで指定した領域にcountバイト数分のデータを読み込み、ファイルポインタを次に読む位置まで進めます。

■ポイント

ファイルの残りのデータがcountに満たない場合や、テキストモードでオープンしたファイルに対する読み込みは、countよりも小さいことがあります。

一度に読み出せる最大バイト数や、テキストモードについてはwriteと同様です。

■例

簡単なtypeコマンドです。

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>

void main(ac, av)
int ac;
char **av;
{
	int fd;
	int nread;
	char buf[512], *p;

	while (--ac) {
		p = *++av;
		if ((fd = open(p, O_RDONLY)) == -1) {
			fprintf(stderr, "cannot open %s¥n", p);
			exit(1);
		}
		while ((nread = read(fd, buf, sizeof buf)) != 0) {
			if (nread == -1) {
				fprintf(stderr, "cannot read %s¥n", p);
				exit(1);
			}
			write(1, buf, nread);
		}
		close(fd);
	}
	exit(0);
}
```

lseek

■書式

```
#include <io.h>
#include <stdio.h>

long lseek (handle, offset, origin);

int handle;      ファイルハンドル
long offset;     バイト数
int origin;      初期位置
```

■戻り値

移動後のファイルポインタの位置が、ファイルの先頭から何バイト離れているかを返します。エラーの場合は−1Lを返し、errnoに次の値を返します。

記号定数	エラー内容
EBADF	ファイルハンドルが不正です。
EINVAL	originの値が不正、もしくはoffsetがファイルの先頭より前を指しています。

■機能

ハンドルで示されるファイルのファイルポインタを移動します。offsetの指定はoriginの指定からの相対位置になります。originで指定する値はstdio.hに定義されています。

記号定数	意味
SEEK_SET	ファイルの先頭
SEEK_CUR	現在のファイルポインタ
SEEK_END	ファイルの終り

■ポイント

ファイルの最後より後へは移動できますが、ファイルの先頭より前に移動するとエラーになります。

■例

与えられたファイルハンドルに対するファイルの先頭10バイトを与えられたバッファに読み込み、ファイルポインタを最初の位置に戻す関数です。途中にエラーが起きたら−1を返します。

```
#include <io.h>
#include <stdio.h>

int seektest(fd, buf)
int fd;
char *buf;
{
        long pos;

        if ((pos = lseek(fd, 0, SEEK_CUR)) == -1L ||
                        lseek(fd, 0, SEEK_SET) == -1L) ||
                        read(fd, buf, 10) == -1 ||
                        lseek(fd, pos, SEEK_SET) == -1L)
                return -1;
        return 0;
}
```

tell

■書式

#include <io.h>

long tell (handle)；

int handle；　　ファイルハンドル

■戻り値

ファイルポインタの位置。−1Lはエラーを示し、errnoに次の値を返します。

記号定数	エラー内容
EBADF	ファイルハンドルが無効。

■機能

指定ファイルハンドルに対応するファイルの現在のファイルポインタの位置を、ファイルの先頭からのバイト数で返します。

■ポイント

シークが不可能なデバイスでは、戻り値は不定となります。

■例

write関数を参照してください。

eof

■書式

#include <io.h>

int eof (handle)；

int handle；　　ファイルハンドル

■戻り値

ファイルの終端なら1、そうでなければ0を返します。エラーが発生したら−1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EBADF	ファイルハンドルが無効です。

■機能

handleで指定したファイルのアクセス位置がファイルの終端(EOF)に達しているかどうかを調べます。

■例

指定ファイルがオープンしただけで既にEOFに達していたら、"NO DATA"と表示する例です。

```
#include <io.h>
#include <fcntl.h>

void main(ac, av)
int ac;
char **av;
{
        int fd;

        if (ac != 2)
                exit(1);
        if ((fd = open(av[1], O_RDONLY)) == -1)
                exit(1);
        if (eof(fd))
                printf("NO DATA¥n");
        close(fd);
}
```

dup
dup2

■書式

```
#include <io.h>

int dup (handle);

int handle;          既にあるファイルハンドル

int dup2 (handle1, handle2);

int handle1;         既にあるファイルハンドル
int handle2;         新しいファイルハンドル
```

■戻り値

dup関数は、新たに作成したファイルハンドルを返します。dup2関数は、成功なら0を返します。どちらもエラーが起った場合は−1を返し、errnoに次の値を返します。

記号定数	エラー内容
EBADF	ファイルハンドルが無効です。
EMFILE	ファイルハンドルが割り当てられません。オープン中のファイルが多すぎます。

■機能

dupおよびdup2関数は、現在オープンされているファイルにふたつ目のファイルハンドルを割り当てます。

dup2関数はhandle2に指定した値のハンドルを強制的にhandle1と同じファイルを参照するハンドルにします。そのときhandle2の値のハンドルが既にオープン中のファイルに対応していたら、handle2に対応したファイルはクローズされます。

■ポイント

割り当てられたファイルハンドルは、元となるファイルハンドルと同じファイルポインタを参照していますから、どちらで読み書きしてもかまいません。

■例

標準出力への出力を、一時的にすべて標準エラー出力に切り変える例です。出力をリダイレクトしても、出力内容は画面に出ることになります。

```
#include <io.h>
#include <stdio.h>

void main()
{
        int savefd, c;

        if ((savefd = dup(1)) == -1)
                exit(1);
        dup2(2, 1);
        while ((c = getchar()) != EOF)
                putchar(c);
        dup2(savefd, 1);
}
```

access

■書式

#include <io.h>

int access (path, mode)；

char ＊path；　パス名
int mode；　アクセスモード

■戻り値

アクセス可能なら0を、そうでなければ−1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	指定されたアクセスモードではアクセスできません。
ENOENT	ファイルまたはディレクトリが存在しません。

■機能

pathで指定されたファイルが、modeで指定されたモードでアクセス可能かどうかを調べます。modeに指定できる値は次の通りです。

値	内容
00	ファイルの存在を調べます。
02	書き込み可能かどうかを調べます。
04	読み出し可能かどうかを調べます。
06	読み書き可能かどうかを調べます。

■ポイント

MS-DOSではディレクトリは常に読み書き可能なため、ディレクトリについては、存在のみを調べます。

■例

chmod関数を参照してください。

chmod

■書式

#include <io.h>

int chmod (path, pmode)；

char ＊path；　ファイルへのパス名
int pmode；　アクセス許可設定

■戻り値

成功なら0を返します。エラーが起きたら−1を返し、errnoには次の値を返します。

記号定数	エラー内容
ENOENT	ファイルが見つかりません。

■機能

pathに指定したファイルへのアクセス許可設定をpmodeに指定した設定に変更します。pmodeに指定する定数はsys/stat.hに定義されています(open関数参照)。

■ポイント

MS-DOSでは読み出し禁止の属性はありませんから、S_IREADを指定しなくても、読み出し禁止にはなりません。

■例

指定ファイルの内、書き込み不許可の属性を持つファイルを書き込み許可に変更するコマンドです。

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>

void main(ac, av)
int ac;
char **av;
{
        while (--ac) {
                char *fn = *++av;
                if (access(fn, 0) == -1)
                        printf("'%s' does not exist¥n", fn);
                else if (access(fn, 2) == -1) {
                        chmod(fn, S_IREAD | S_IWRITE);
                        printf("'%s' mode changed¥n", fn);
                }
        }
        exit(0);
}
```

chsize

■書式

#include <**io.h**>

int chsize (handle, size)；

int handle；　ファイルハンドル
long size；　ファイルサイズ

■戻り値

成功した場合は0、エラーが起きた時は－1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCESS	指定ファイルはロックされています。(MS-DOSバージョン3.0以降のみ)
EBADF	ファイルハンドルが無効、または書き込み禁止です。
ENOSPC	デバイスに空き領域がありません。

■機能

handleに対応するファイルのファイルサイズを変更します。指定するファイルは、書き込み可能でオープンされている必要があります。

■ポイント

サイズを広げた場合は、広げた部分をNULL（'¥0'）で初期化し、縮小した場合は、縮小部分のデータは失われます。

■例

指定ファイルに対し、サイズが指定されればサイズを変更し、指定されなければ、そのファイルのサイズを表示するコマンドです。

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <stdlib.h>

void main(ac, av)
int ac;
char **av;
{
        long sz, l;
        char *p;

        if (!--ac) {
                fprintf(stderr, "chsize [-<size>] filename ..¥n");
                exit(1);
        }
```

```
	if (*(p = *++av) == '-') {
		sz = atol(++p);
		av++;
		ac--;
	} else
		sz = -1L;
	while (ac--) {
		int fd;
		char *fn = *av++;

		if ((fd = open(fn, O_RDWR | O_BINARY)) == -1) {
			fprintf(stderr, "cannot open %s¥n", fn);
			exit(1);
		}
		if (sz == -1L) {
			if ((l = filelength(fd)) == -1) {
				fprintf(stderr,
					"cannot get filesize '%s'¥n", fn);
				exit(1);
			}
			printf("%s: %10ld¥n", fn, filelength(fd));
		} else if (chsize(fd, sz) != 0) {
			fprintf(stderr, "cannot change size '%s'¥n", fn);
			exit(1);
		}
		close(fd);
	}
	exit(0);
}
```

filelength

■書式

#include <io.h>

long filelength (handle)；

int handle；　　ファイルハンドル

■戻り値

　ファイルの長さをバイト数で返します。エラーが起ったら－1Lを返し、errnoにEBADF(ファイルハンドルが無効)を設定します。

■機能

　handleで指定したファイルの大きさを調べます。

■例

　chsize関数を参照してください。

fstat

■書式

＃include <**sys/types.h**>
＃include <**sys/stat.h**>

int fstat (handle, buffer)；

int handle；　　　　ファイルハンドル
struct stat ＊buffer；　格納域へのポインタ

■戻り値

正常なら0、エラーなら−1を返し、errnoにEBADF(ファイルハンドルが無効)を返します。

■機能

handleで指定されたオープン中のファイルに関する情報をbufferで指定されたstat型の構造体に設定します。
stat型の構造体はsys/stat.hに定義されており、次に示すフィールドを持ちます。

メンバ	内容
st_atime	ファイルを最後に更新した日時。
st_ctime	st_atimeと同じ。
st_dev	ファイルが格納されているドライブ番号。デバイスなら、handleの値。
st_mode	ファイルモード。
st_mtime	st_atimeと同じ。
st_nlink	常に1。
st_rdev	st_devと同じ。
st_size	ファイルの大きさ(バイト数)。

■ポイント

プロテクトモードでは、st_dev, st_rdevの値は定義されていません。

■例

ファイル名を指定すると、ドライブ／サイズ／時間を表示するプログラムです。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <io.h>
#include <time.h>

void main(ac, av)
int ac;
char **av;
{
        int fd;
        char *p;
        struct stat sbuf;

        while (--ac) {
                p = *++av;
                if ((fd = open(p, O_RDONLY)) == -1) {
                        fprintf(stderr, "cannot open '%s'¥n", p);
                        exit(1);
                }
                if (fstat(fd, &sbuf) == -1) {
                        fprintf(stderr, "cannot get stat '%s'¥n", p);
                        exit(1);
                }
                printf("name:   %s¥n", p);
                printf("drive:  %d¥n", sbuf.st_dev);
                printf("size:   %ld¥n", sbuf.st_size);
                printf("time:   %s¥n", ctime(&sbuf.st_atime));
                close(fd);
        }
        exit(0);
}
```

isatty

■書式

#include <io.h>

int isatty (handle)；

int handle；　　デバイスへのハンドル

■戻り値

指定デバイスがキャラクタデバイスなら0以外、それでなければ0を返します。

■機能

handleで指定したデバイスがキャラクタデバイスであるかどうかを調べます。

■例

標準入出力のデバイスタイプを表示する例です。リダイレクトの方法によって、結果が変わるのがわかります。

```
#include <io.h>
#include <stdio.h>

char *devtype(fd)
int fd;
{
        return isatty(fd) ? "character-device" : "block-device";
}

void main()
{
        fprintf(stderr, " stdin:  %s¥n stdout: %s¥n",
                                devtype(0), devtype(1));
        exit(0);
}
```

locking

■書式

```
#include <sys/locking.h>
#include <io.h>

int locking (handle, mode, nbyte);

int handle;      ファイルハンドル
int mode;        ロックモード
long nbyte;      ロックするバイト数
```

■戻り値

正常なら0、エラーが起きたら−1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	既にロックまたはロック解除されています。
EBADF	ファイルハンドルが不正です。
EDEADLOCK	ロックが不正です。10回の再試行にも関わらず、ロックできなかった場合に返されます。
EINVAL	引数が間違っています。

■機能

handleで指定したファイルの現在のファイルポインタから、nbyteで指定したバイト数の範囲をロックしたりロック解除したりします。

modeで指定する定数はsys/locking.hに次の意味で定義されています。

記号定数	意味
LK_LOCK	指定した範囲をロックします。ロックできなければ、1秒後に再度試み、10回の再試行でも失敗したらエラーとなります。
LK_RLCK	LK_LOCKと同じです。
LK_NBLCK	指定範囲をロックします。できなければエラーとなります。
LK_NBRLCK	LK_NBLCKと同じです。
LK_UNLCK	指定範囲のロックを解除します。

ひとつのファイル中、複数の部分をロックできますが、範囲が重なってはできません。

ロックしていない部分をロック解除してはなりません。複数のロック部分を一度には解除できません。部分ごとに行ってください。

ファイルをクローズする前に必ずロック解除をしてください。

■ポイント

MS-DOSのバージョン3.xのネットワークで、SHAREコマンドを組み込んで使用します。OS/2では必要ありません。

■例

渡されたファイルハンドルの先頭から現在のファイルポインタの位置までをロックする例です。

```
#include <stdio.h>
#include <io.h>
#include <sys/locking.h>

int lockit(fd)
int fd;
{
        long l = tell(fd);
        int r;

        lseek(fd, 0L, SEEK_SET);
        r = locking(fd, LK_NBLCK, l);
        lseek(fd, l, SEEK_SET);
        return r;
}
```

_makepath ver 5

■書式

#include <stdlib.h>

void _makepath (path, drive, dir, fname, ext);

char *path; 生成されたpath名
char *drive; ドライブ名
char *dir; ディレクトリ名
char *fname; ファイル名
char *ext; 拡張子名

■戻り値

変換した結果をpathに返します。戻り値はありません。

■機能

_makepath関数は指定されたdrive, dir, fname, extより、フルパス名を生成します。それぞれの引数には以下の内容をセットします。

引数	内容
drive	ドライブ名です。"a:"や"C"のように指定します。コロンはなくてもかまいません。NULLを指定した場合は、生成された結果にドライブ名が含まれません。
dir	ディレクトリ名です。"¥¥usr¥¥bin¥¥src"や、"/usr/bin/"のように指定します。文字列の最後の円記号(¥)やスラッシュ(/)はなくてもかまいません。NULLを指定した場合は、生成された結果にディレクトリ名が含まれません。
fname	ファイル名です。拡張子は含めません。
ext	拡張子です。".c"や"DOC"のように指定します。先頭の'.'はなくてもかまいません。NULLを指定した場合は、生成された結果には'.'を含む拡張子が含まれません。

■ポイント

それぞれの引数の長さに制限はありませんが、生成された結果が_MAX_PATHを越えるような引数を指定してはいけません。

■例

ドライブ名、ディレクトリ名、ファイル名、拡張子を指定すると、結合した結果を出力します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        char path[_MAX_PATH];

        if (argc != 5) {
                fputs("ドライブ名 ディレクトリ名 "¥
                        "ファイル名,拡張子を結合します", stderr);
                return 2;
        }
        _makepath(path, argv[1], argv[2], argv[3], argv[4]);
        puts(path);
        return 0;
}
```

mktemp

■書式

#include <io.h>

char *mktemp (template);

char *template;　　　　ファイル名のパターン

■戻り値

ファイル名へのポインタを返します。
templateの書式が間違っていたり、これ以上名前が作れない場合はNULLを返します。

■機能

与えられたtemplateを元にして、名前が重複しないようなファイル名を作ります。
templateは次のような書式にします。

baseXXXXXX

baseはユーザが指定する部分で、6つのXはmktemp関数により、最初のXは英数字に、他の5つは数字に置き換えられます。最初の英数字は、最初に呼び出された時には〝0″が、以降は〝a″から〝z″のアルファベットが使用されます。

■ポイント

前回返した名前でファイルが作られなかった場合は同じ名前を返します。

■例

テンポラリファイルを作成します。

```
#include <io.h>
#include <stdio.h>

char *template = "tmXXXXXX";

main()
{
        FILE *fp;
        char *p;

        if ((p = mktemp(template)) == NULL) {
                printf("cannot make tempfile¥n");
                exit(1);
        }
        if ((fp = fopen(p, "w")) == NULL) {
                printf("cannot open tempfile¥n");
                exit(1);
        }
        /*
                一時ファイルを使用する処理
        */
        fclose(fp);
}
```

remove　ver 4

■書式

＃include ＜io.h＞
＃include ＜stdio.h＞

int remove (name)；

const char ＊name；　　ファイルネーム

■戻り値

ファイルが正常に削除された場合は0を返します。戻り値が−1の場合はエラーが発生したことを示し、errnoに以下の記号定数がセットされます。

記号定数	意味
EACCES	nameで指定されたファイルが読み込み専用である。
ENOENT	nameで指定されたファイルが存在しない。

■機能

remove関数はnameで指定されたファイルを削除します。

■ポイント

remove関数はstdio.hの他にio.hでも定義されています。
ディレクトリの削除はできません。

■例

コマンドラインに指定されたファイルを削除します。

```
#include <stdio.h>

main(int argc, char **argv)
{
        int i;

        if (argc == 1) {
                printf("削除したいファイル名を指定してください。¥n");
                exit(1);
        }
        for (i = 1; i < argc; i++) {
                printf("%15s ... ", argv[i]);
                if (remove(argv[i]) != 0) {
                        switch (errno) {
                                case EACCES:
                                        printf("Read only...skip this file¥n");
                                        break;
                                case ENOENT:
```

```
                                        printf("Can¥'t find¥n");
                                        break;
                                default:
                                        printf("Unknown error¥n");
                                        break;
                        }
                        exit(1);
                } else
                        printf("Deleted¥n");
        }
        exit(0);
}
```

rename

■書式

```
#include <io.h>
#include <stdio.h>

int rename (oldname, newname);

char *oldname;        前の名前
char *newname;        新しい名前
```

■戻り値

成功なら0、エラーの場合は0以外の値を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	newnameによって指定したファイルが既に存在するか、作成できない。 または、oldnameがディレクトリであって、newnameが異なるパスを指定している。
ENOENT	oldnameで指定したファイルまたはパス名が存在しない。
EXDEV	ファイルを異なるドライブまたはデバイスへ移そうとした。

■機能

rename関数はoldで指定されたファイル名をnewで指定されたファイル名に変更します。

oldとnewでパスを変えることによって、oldが存在するディレクトリを移動することができます。たとえば、

```
rename ("test", "¥usr¥bin¥test");
```

では、カレントディレクトリのtestを¥usr¥binに移動します。

oldにはディレクトリ名を指定することができますが、ディレクトリの移動は不可能です。newとoldではパス名を一致させます。ディレクトリを移動させようとした場合はエラーとなり、errnoにEACCESが返されます。

■ポイント

newに既に存在しているファイルまたはディレクトリの名前を指定するとエラーになります。また、newとoldで指定されているドライブが違う場合もエラーとなります。

MS-Cのver4.0より以前のバージョンでは、引数newとoldの順序が逆になっています。注意してください。

■例

簡単なrenameコマンドです。

```
#include <io.h>
#include <stdio.h>

void main(ac, av)
int ac;
char **av;
{
	if (ac != 3)
		exit(1);
	if (rename(av[1], av[2]) != 0) {
		printf("cannot rename %s to %s¥n", av[1], av[2]);
		exit(1);
	}
	exit(0);
}
```

setmode

■書式

#include <fcntl.h>
#include <io.h>

int setmode (handle, mode);

int handle;　　ファイルハンドル
int mode;　　変換モード

■戻り値

以前のモードを返します。−1はエラーを示し、errnoに次の値をセットします。

記号定数	エラー内容
EBADF	ファイルハンドルが無効
EINVAL	引数modeが無効

■機能

handleで指定するファイルの変換モードをmodeで指定されたモードに設定します。modeは、次の値のいずれかで、fcntl.hに定義されています。

記号定数	意味
O_TEXT	テキストモード
O_BINARY	バイナリモード

■ポイント

普通はstdin, stdout, stderr, stdaux, stdprnのデフォルトの変換モードの変更に使用しますが、通常のファイルに対しても使用できます。

■例

stdoutを、引数が0ならテキストモード、0以外ならバイナリモードに変更する関数です。

```
void setstdout(f)
int f;
{
        setmode(fileno(stdout), f ? O_BINARY : O_TEXT);
}
```

_splitpath　ver 5

■書式

#include <stdlib.h>

void _splitpath (path, drive, dir, fname, ext);

char *path;　解析するフルパス名
char *drive;　ドライブ名
char *dir;　ディレクトリ名
char *fname;　ファイル名
char *ext;　拡張子名

■戻り値

変換した結果をdrive, dir, fname, extに設定する以外は、戻り値はありません。

■機能

_splitpath関数は、フルパス名pathを解析し、4要素に分解します。

引数	内容
drive	pathにドライブ名のフィールドが存在する場合は、"C:"のようにコロンまでがセットされます。存在しない場合は空文字列がセットされます。
dir	pathにディレクトリ名のフィールドが存在する場合は、"/USR/"のように最後にスラッシュ(/)を付加してセットします。'/'と'¥'が混在する場合があります。存在しない場合は空文字列がセットされます。
fname	ファイル名です。拡張子は含めません。
ext	拡張子です。拡張子が存在すれば、".DOC"のようにピリオドから拡張子がセットされます。存在しない場合は空文字列がセットされます。

■ポイント

drive, dir, fname, extに必要な長さは、_MAX_DRIVE, _MAX_DIR, _MAX_FNAME, _MAX_EXTとstdlib.hで定義されています。

■例

プロセス名argv[0]を解析し表示します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        char drive[_MAX_DRIVE], dir[_MAX_DIR], fname[_MAX_FNAME];
        char ext[_MAX_EXT];

        _splitpath(argv[0], drive, dir, fname, ext);
        puts(argv[0]);
        puts(drive);
        puts(dir);
        puts(fname);
        puts(ext);
        return 0;
}
```

stat

■書式

#include <sys/types.h>
#include <sys/stat.h>

int stat (path, buffer);

char *path;　　　　パス名
struct stat *buffer;　格納域へのポインタ

■戻り値

正常なら0、エラーなら−1を返し、errnoにENOENT(ファイルまたはディレクトリが見つからない)を返します。

■機能

pathで指定されたファイルやディレクトリの情報を得て、bufferで指定されたstat構造体にセットします。
stat構造体についてはfstat関数を参照してください。

■ポイント

stat構造体には、他の処理系との互換性のために、MS-DOSでは使用しないフィールドも定義されています。

■例

ファイル名を指定すると、ドライブ/サイズ/時間を表示するプログラムです。fstat関数と違い、オープンしていないファイルに対して、直接情報を得ることができます。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>

void main(ac, av)
int ac;
char **av;
{
        int fd;
        char *p;
        struct stat sbuf;

        while (--ac) {
                p = *++av;
                if (stat(p, &sbuf) == -1) {
                        fprintf(stderr, "cannot get stat '%s'¥n", p);
                        exit(1);
                }
                printf("name:   %s¥n", p);
                printf("drive:  %d¥n", sbuf.st_dev);
                printf("size:   %ld¥n", sbuf.st_size);
                printf("time:   %s¥n", ctime(&sbuf.st_atime));
        }
        exit(0);
}
```

umask

■書式

#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

int umask（pmode）；

int pmode；　　アクセスマスク

■戻り値

以前のpmodeの値を返します。アクセスビットはsys/stat.hに次のように定義されています。

記号定数	意味
S_IWRITE	書き込み可能
S_IREAD	読み出し可能

■機能

そのプロセスで新しく作成するファイルへのアクセス許可をマスクします。つまりpmodeで指定したビットの1の部分がアクセス不許可になります。

■例

この関数を呼び出した後に作成されるファイルは読み出し専用となります。

```
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

void readonly()
{
        umask(S_IWRITE);
}
```

unlink

■書式

#include <**io.h**>
#include <**stdio.h**>

int unlink (path);

const char *path;　　　削除するファイルのパス名

■戻り値

ファイルが削除できた場合は0を返します。−1はエラーを示し、errnoに次の値を返します。

記号定数	エラー内容
EACCES	パス名がディレクトリか、書き込み禁止のファイルを指定しています。
ENOENT	パス名で指定されたファイルが存在しません。

■機能

pathで指定されたファイルを削除します。

■例

tempというファイルを削除します。

```
main()
{
        static char filename[] = "temp";

        if (unlink(filename) == -1)
                printf("cannot remove %s¥n", filename);
        else
                printf("removed %s¥n", filename);
}
```

bdos

■書式

```
#include <dos.h>

int bdos (function, dx, al);

int funciton;
unsigned int dx;        DXレジスタの値
unsigned int al;        ALレジスタの値
```

■戻り値

AXレジスタの値を返します。

■機能

AHレジスタにfunction、ALレジスタにal、DXレジスタにdxを設定し、MS-DOSファンクションコールを行います。

_chain_intr　　ver 5

■書式

```
#include <dos.h>

void _chain_intr (handler);

void (interrupt far *handler) ();
```

■戻り値

ありません。

■機能

割り込みハンドラから別の割り込みハンドラを起動します。

_disable
_enable

ver 5

■書式

＃include ＜dos.h＞

void disable (void);
void enable (void);

■戻り値

ありません。

■機能

_disable関数はハードウェア割り込みを禁止します。
_enable関数はハードウェア割り込みを許可します。

■ポイント

_disable関数は8086ニーモニックのcli、_enable関数はstiに相当します。

_dos_allocmem　ver 5

■書式

```
#include <dos.h>

unsigned _dos_allocmem (size, segment);

unsigned size;
unsigned *segment;
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返し、errnoにENOMEMを設定します。

■機能

MS-DOSファンクションコールによりメモリブロックを割り当てます。
sizeパラグラフを割り当て、割り当てられたメモリブロックのセグメントアドレスをunsigned型ポインタsegmentに格納します。

■ポイント

割り当てられたメモリブロックは必ずパラグラフ境界から始まります。1パラグラフは16バイトですから、割り当てたいバイト数の16分の1の数値をsizeとして与えます。割り当てられた領域を解放するためには、_dos_freemem()を使用しなければなりません。

_dos_close　ver 5

■書式

#include <dos.h>

unsigned _dos_close (handle);

int handle;　　ファイルハンドル

■戻り値

成功した場合0を返し、エラーが発生した場合は、MS-DOSのエラーコードを返します。errnoにはEBADFがセットされます。

■機能

MS-DOSファンクションコールを用いてファイルクローズを行います。

■ポイント

主として_dos_openによりオープンされたファイルのクローズに用います。

_dos_creat
_dos_creatnew

ver 5

■書式

```
#include <dos.h>

unsigned _dos_creat (path, attr, handle);
unsigned _dos_creatnew (path, attr, handle);

char *path;         ファイルのパス名
unsigned attr;      ファイルの属性
int *handle;        ファイルハンドル
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。errnoにエラー理由を示す数値が代入されます。

■機能

MS-DOSファンクションコールを用いてパス名path、ファイル属性attrの新しいファイルをオープンし、ファイルハンドルを*handleに返します。オープンしようとするファイルが既に存在する場合、_dos_creat()関数は既存ファイルの内容を破棄します。_dos_creatnew()関数ではエラーとなりオープンできません。

■ポイント

ファイル属性については_dos_setfileattrを参照してください。

dosexterr

■書式

```
#include <dos.h>

int dosexterr (buffer)

struct DOSERROR *buffer;        構造体へのポインタ
```

■戻り値

MS-DOS拡張エラー番号が返されます。

■機能

MS-DOSファンクションコール0x59を実行し、拡張エラー情報をbufferに返します。bufferは構造体DOSERRORへのポインタで、構造体DOSERRORはdos.hで次のように定義されています。

```
struct DOSERROR {
    int exterror;
    char class;
    char action;
    char locus;
};
```

exterrorには、拡張エラーコードが返されます。classにはエラークラス、actionには対処可能な処理、locusには付加情報が格納されます。詳しくは、MS-DOSプログラマーズリファレンスマニュアルを参照してください。

_dos_findfirst
_dos_findnext

ver 5

■書式

unsigned _dos_findfirst (path, attr, buffer)；
unsigned _dos_findnext (buffer)；

char ＊path；	検索するファイルのパス名
unsigned attr；	検索するファイルの属性
struct find_t ＊buffer；	構造体を指すポインタ

■戻り値

検索に成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

_dos_findfirst()関数はパス名path、属性attrに最初に一致するディレクトリエントリを検索します。

検索されたファイルの情報は、bufferに返されます。bufferは構造体find_t型へのポインタで、dos.hで定義されています。構造体find_tは以下ようなフォーマットを持っています。

```
struct find_t  {
     char reserved [21]；
     char attrib；
     unsigned wr_time；
     unsigned wr_date；
     long size；
     char name [13]；
}；
```

attribにはファイルの属性、wr_date, wr_timeには最終書き込み日時が、sizeにはファイルサイズ、nameには検索されたファイル名が返されます。

reservedはシステム予約領域なので変更してはいけません。

_dos_findnext()関数は、次のディレクトリエントリを検索します。bufferには_dos_findfirst()関数が返した構造体find_tへのポインタを与え、与えられたbufferが最初に検索された時と同じ条件で次のファイルを検索します。

■ポイント

属性指定に関わらず、通常ファイルは必ず検索対象となります。

ファイル属性については_dos_setfileattrを参照してください。

■例

カレントディレクトリにあるすべてのファイル名を表示します。

```
#include <dos.h>
#include <stdio.h>

main()
{
        int r;
        struct find_t fb;

        r = _dos_findfirst("*.*",
                -1, &fb);
        while (r == 0) {
                puts(fb.name);
                r = _dos_findnext(&fb);
        }
}
```

_dos_freemem　　ver 5

■書式

#include <dos.h>

unsigned _dos_freemem (segment);

unsigned segment;　　解放するブロック

■戻り値

成功した場合0を返し、エラーが発生した場合は、MS-DOSのエラーコードを返し、errnoにENOMEMを設定します。

■機能

_dos_allocmem()により割り当てられたメモリブロックを解放します。

■ポイント

引数segmentには、_dos_allocmem()により得られたパラグラフ番号を入れなければなりません。

_dos_getdate　ver 5

■書式

```
#include <dos.h>

void _dos_getdate (date)

struct dosdate_t *date;        構造体へのポインタ
```

■戻り値

ありません。

■機能

システムクロックから現在の日付を得ます。結果は、dateに返されます。dateは構造体dosdate_tへのポインタで、構造体dosdate_tはdos.hで以下のように定義されています。

```
struct dosdate_t {
    unsigned char day;
    unsigned char month;
    unsigned int year;
    unsigned char dayofweek;
};
```

dayには1から31の値が返され日を示します。monthには1から12の値が返され月を示します。yearには1980から2099の値が返され年を示します。dayofweekは0から6までの数値が返され、0は日曜、1は月曜、6が土曜を意味します。

■ポイント

time.hで宣言されている標準の時刻関数等は、UNIXとの互換性のために多くの付加機能が付いています。_dos_getdate()関数はMS-DOSファンクションコールによって得られた日付情報をほぼそのまま返すため、サイズが小さくなります。

互換性を重視する場合は標準の時刻関数、サイズや実行速度を重視する場合は_dos_getdate等を使用すべきでしょう。

■例

日曜なら、"Today is Sunday!!" と表示します。

```
#include <conio.h>
#include <dos.h>

void main(void)
{
        struct dosdate_t date;

        _dos_getdate(&date);
        if (date.dayofweek == 0)
                cputs("Today is Sunday!!¥r¥n");
}
```

_dos_getdiskfree　ver 5

■書式

```
#include <dos.h>

unsigned _dos_getdiskfree (drive, dsikspec);

unsigned drive;                    ドライブ番号
struct diskfree_t  *diskspec;      構造体へのポインタ
```

■戻り値

成功した場合0を返し、エラーが発生した場合は0以外の値を返します。

■機能

指定されたディスクの情報を得ます。driveにはドライブ番号を与え、結果をdiskspecに返します。

driveの値が0のときはカレントドライブ、1はA:、2はB:のように対応しています。

diskspecは構造体diskfree_tへのポインタで、構造体diskfree_tはdos.hで定義されています。

```
struct diskfree_t  {
    unsigned total_clusters;
    unsigned avail_clusters;
    unsigned sectors_per_cluster;
    unsigned bytes_per_sector;
};
```

total_clustersはドライブの全クラスタ数が返されます。avail_clustersは使用可能な未使用のクラスタ数が返されます。

sectors_per_clusterは1クラスタに含まれるセクタ数、bytes_per_sectorには1セクタに含まれるバイト数が返されます。

■ポイント

全ディスク容量は、

total_clusters * sectors_per_cluster * bytes_per_sector で求められます。

空きディスク容量は、

avail_clusters * sectors_per_cluster * bytes_per_sector で求められます。

■例

カレントドライブのディスク使用状況を表示します。

```
#include <stdio.h>
#include <dos.h>

main()
{
        struct diskfree_t diskspec;
        long total, used, free;
        long bpc;
        int percent;

        _dos_getdiskfree(0, &diskspec);
        bpc = (long)diskspec.sectors_per_cluster
                * (long)diskspec.bytes_per_sector;
        total = (long)diskspec.total_clusters * bpc;
        free = (long)diskspec.avail_clusters * bpc;
        used = total - free;
        percent = (int)(100L * used / total);

        printf("%9ld bytes free (%d%% used)¥n", free, percent);
}
```

_dos_getdrive　ver 5

■書式

#include <dos.h>

void _dos_getdrive (drive);

unsigned *drive;　　ドライブ番号

■戻り値

ありません。

■機能

カレントドライブ番号を取得し、driveに返します。A:の場合は1、B:は2のように対応しています。

■例

カレントドライブ名を表示します。

```
#include <stdio.h>
#include <dos.h>

main()
{
        unsigned drive;
        _dos_getdrive(&drive);
        printf("current drive is %c:¥n",
                drive + '@');
}
```

_dos_getfileattr　ver 5

■書式
```
#include <dos.h>

unsigned _dos_getfileattr (path, attr);

char *path;          フルパス名
unsigned *attr;      ファイル属性
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

pathで指定されたファイル・ディレクトリの属性を得てattrに返します。

■ポイント

ファイル属性については_dos_setfileattrを参照してください。

_dos_getftime　　ver 5

■書式

```
#include <dos.h>

unsigned _dos_getftime (handle, date, time);

int handle;          ファイルハンドル
unsigned *date;      日付
unsigned *time;      時刻
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

オープンされているファイルhandleの最終更新日時をdate, timeに得ます。

■ポイント

date, timeは、MS-DOSの時刻フォーマットで格納されます。

_dos_gettime　ver 5

■書式

#include <dos.h>

void _dos_gettime (time) ;

struct dostime_t * time ;　　構造体へのポインタ

■戻り値

ありません。

■機能

MS-DOSファンクションコールにより現在時刻を取得して、timeに返します。timeは構造体dostime_tへのポインタで、dostime_tはdos.hで次のように定義されています。

```
struct dostime_t {
    unsigned char hour ;
    unsigned char minute ;
    unsigned char second ;
    unsigned char hsecond ;
} ;
```

hourには時、minuteには分、secondには秒、hsecondには100分の1秒が格納されます。

■ポイント

_dos_getdate()関数を参照してください。

_dos_getvect　　ver 5

■書式
#include <dos.h>

void (interrupt far * _dos_getvect (vect))() ;

unsigned vect ;　　割り込み番号

■戻り値
割込型関数へのfarポインタを返します。

■機能
割り込みベクタvectの内容を得ます。

_dos_keep　　ver 5

■書式
#include <dos.h>

void _dos_keep (retcode, memsize)

unsigned retcode ;　　終了ステータスコード
unsigned memsize ;　　常駐するために割りあてられるパラグラフ数

■戻り値
ありません。

■機能
プログラムを常駐終了させます。retcodeはリターンコードとしてDOSに返されます。常駐するメモリのパラグラフサイズをmemsizeに与えます。

■ポイント
1パラグラフは16バイトです。

_dos_open　ver 5

■書式

```
#include <dos.h>
#include <fcntl.h>
#include <share.h>

unsigned _dos_open (path, mode, handle)

char *path;            ファイルのパス名
unsigned mode;         アクセスモード
int *handle;           ファイルハンドル
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

MS-DOSファンクションコールを用いて、ファイルpathをモードmodeでオープンし、ファイルハンドルをhandleに返します。modeには以下の定数を ｜ で組み合わせて指定します。

記号定数	内容
O_RDONLY	読み出し専用
O_WRONLY	書き込み専用
O_RDWR	読み書き両用
SH_COMPAT	互換モード
SH_DENYRW	読み書き拒否
SH_DENYWR	書き込み拒否
SH_DENYRD	読み出し拒否
SH_DENYNONE	読み書き許可
O_NOINHERIT	子プロセスに継承しない

■ポイント

SHで始まるモードはシェアリングモードに関する設定です。これらの機能を使用するためにはSHARE.EXEが実行されていなければなりません。

open()関数やfopen()関数のようなテキストモード・バイナリモードの区別はありません。

_dos_read　ver 5

■書式

```
#include <dos.h>

int _dos_read (handle, buffer, count, bytes);

int handle;              ファイルハンドル
void far *buffer;        バッファ
unsigned count;          読み込むバイト数
unsigned *bytes;         実際に読み込んだバイト数
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

MS-DOSファンクションコールを用いて、ファイルhandleからcountバイト読み込み、bufferに格納します。実際に読み込まれたバイト数をbytesに返します。

■ポイント

改行コードの変換は行われません。bufferはfarポインタです。

_dos_setblock ver 5

■書式

```
#include <dos.h>

unsigned _dos_setblock (size, seg, max);

unsigned size;          メモリブロックサイズ
unsigned seg;           セグメントディスクリプタ
unsigned *max;          最大メモリブロックサイズ
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

MS-DOSのメモリアロケートにより割り当てられたメモリブロックsegの大きさをsizeパラグラフに変更します。変更できなかった場合には、最大のメモリブロックのサイズをmaxに返します。

_dos_setdate ver 5

■書式

```
#include <dos.h>

unsinged _dos_setdate (date);

struct dosdate_t *date;        構造体へのポインタ
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

現在のシステム日付をdateに変更します。dateは構造体dosdate_tへのポインタで、構造体dosdate_tはdos.hで定義されています。詳細については_dos_getdate()関数を参照してください。

_dos_setdrive　ver 5

■書式

#include <dos.h>

void _dos_setdrive (drive, drives);

unsigned drive;　　ドライブ名
unsigned *drives;　　使用可能なドライブ数

■戻り値

ありません。

■機能

カレントドライブをdriveに変更します。接続されているドライブ数をdrivesに返します。driveは1以上の数値で、A:は1、B:は2のように対応しています。

_dos_setfileattr　ver 5

■書式

```
#include <dos.h>

unsigned _dos_setfileattr (path, attr);

char *path;        パス名
unsigned attr;     ファイル属性
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

ファイルpathの属性をattrに変更します。attrには以下の定数を指定します。

記号定数	内容
_A_NORMAL	通常ファイル
_A_RDONLY	読み出し専用ファイル
_A_HIDDEN	隠しファイル
_A_SYSYTEM	システムファイル
_A_VOLID	ボリュームID
_A_SUBDIR	サブディレクトリ
_A_ARCH	アーカイブ属性

_dos_setftime ver 5

■書式
#include <dos.h>

unsigned _dos_setftime (handle, date, time);

int handle; ファイルハンドル
unsigned date; 日付
unsigned time; 時刻

■戻り値
成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能
オープンされているファイルhandleの最終更新日時をdate, timeに変更します。date, timeはMS-DOSの日時フォーマットで指定します。

_dos_settime ver 5

■書式
#include <dos.h>

unsigned _dos_settime (time);

struct dostime_t *time; 構造体へのポインタ

■戻り値
成功した場合0を返し、エラーが発生した場合は0以外の値を返します。

■機能
現在のシステム時刻をtimeに変更します。timeは構造体dostime_tへのポインタで、dostime_tはdos.hで定義されています。詳しくは_dos_gettime関数を参照してください。

_dos_setvect　ver 5

■書式

```
#include <dos.h>

void _dos_setvect (vect, handler);

unsigned vect;                             割り込み番号
void (interrupt far *handler) ();          割り込みハンドラ
```

■戻り値

ありません。

■機能

割込ベクタ番号vectの処理アドレスをhandlerに変更します。

■ポイント

handlerはinterrupt型の関数へのポインタでなければなりません。

_dos_write　ver 5

■書式

```
#include <dos.h>

unsigned _dos_write (handle, buffer, count, bytes);

int handle;             ファイルハンドル
void far *buffer;       バッファ
unsigned count;         書き込むバイト数
unsigned *bytes;        実際に書き込んだバイト数
```

■戻り値

成功した場合0を返し、エラーが発生した場合はMS-DOSのエラーコードを返します。

■機能

MS-DOSファンクションコールを用いて、ファイルhandleにbufferからcountバイト書き込みます。実際に書き込まれたバイト数をbytesに返します。

■ポイント

_dos_read関数同様、改行コードの変換は行われません。
bufferはfarポインタなので注意が必要です。

FP_SEG
FP_OFF

■書式

#include 〈dos.h〉

unsigned FP_SEG (fp);
unsigned FP_OFF (fp);

void far *fp;　　　メモリへのポインタ

■戻り値

得られた値を返します。

■機能

FP_SEGはfarポインタのセグメント部分の値を得ます。
FP_SEGはfarポインタのオフセット部分の値を得ます。

■ポイント

これらはマクロで定義されています。FP_OFF()や、FP_SEG()は左辺値として使用することができます。

■例

9801のグラフィック画面の青プレーンを消去します。

```
#include <dos.h>

main()
{
        char far *p;
        int i;

        FP_SEG(p) = 0xa800;
        FP_OFF(p) = 0;
        for (i = 0; i < 32000; i++)
                *p++ = 0;
}
```

_harderr
_hardresume
_hardretn　　ver 5

■書式

```
#include <dos.h>

void _harderr (handler);
void _hardresume (result);
void _hardretn (error);

void (far *handler) ();    新しいハンドラ
int result;                引数
int error;                 エラーコード
```

■戻り値

ありません。

■機能

_harderrはハードウェアエラーが発生した場合に呼び出される関数をhandlerにします。handlerは次のような関数でなくてはなりません。

```
void handler (deverror, errcode, devhdr);
unsigned deverror;
unsigned errcode;
unsigned far *devhdr;
```

ハードウェア割り込みが発生すると、deverrorにデバイスエラーコード、errcodeはエラーコード、devhdrはデバイスヘッダへのfarポインタを与えてhandlerが呼び出されます。

_hardresume関数はhandlerの中で用いられ、処理をMS-DOSに戻すことができます。resultには次のような定数を与え、これらの定数はdos.hで定義されています。

記号定数	内容
_HARDERR_IGNORE	無視
_HARDERR_RETRY	再試行
_HARDERR_ABORT	中断
_HARDERR_FAIL	失敗

_hardretn関数はhandlerの中で用いられ、処理をアプリケーションに戻すことができます。errorはMS-DOSのエラーコードを与えます。

int86　intdos
int86x　intdosx

■書式

```
#include <dos.h>

int int86 (vect, inregs, outregs);
int int86x (vect, inregs, outregs, segregs);
int intdos (inregs, outregs);
int intdosx (inregs, outregs, segregs);

int vect;                  割り込み番号
union REGS inregs;         共用体へのポインタ
union REGS outregs;        共用体へのポインタ
struct SREGS sregs;        構造体へのポインタ
```

■戻り値

AXレジスタの内容を返します。

■機能

これらの関数はソフトウェア割り込みを発生させます。int86関数、int86x関数では割込番号vectを、intdos関数、intdosx関数では割込番号0x21（MS-DOSファンクションコール）を呼び出します。

呼び出すときのレジスタの内容は、inregsに与えます。割り込みが終了した後のレジスタの状態はoutregsに返されます。inregs、outregsは共用体REGS型へのポインタで、共用体REGSはdos.hで次のように定義されています。

```
struct WORDREGS {
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
    unsigned int si;
    unsigned int di;
    unsigned int cflag;
};
```

```
struct BYTEREGS {
    unsigned char al, ah ;
    unsigned char bl, bh ;
    unsigned char cl, ch ;
    unsigned char dl, dh ;
} ;

union REGS {
    struct WORDREGS x ;
    struct BYTEREGS h ;
} ;
```

int86x関数、intdosx関数ではセグメントレジスタの内容segregsも受け渡し可能です。segregsは構造体SREG型へのポインタで、構造体SREGSはdos.hで次のように定義されています。

```
struct SREGS {
    unsigned int es ;
    unsigned int cs ;
    unsigned int ss ;
    unsigned int ds ;
} ;
```

segread

■書式

\#include ＜dos.h＞

void segread（segregs)

struct SREGS ＊segregs；　　構造体へのポインタ

■戻り値

ありません。

■機能

現在のセグメントレジスタの内容をsegregsに返します。segregsは構造体SREGSへのポインタです。構造体SREGSについてはint86x関数を参照してください。

■例

プログラムが実行されたときのCSレジスタの値を16進数で表示します。

```
#include <dos.h>
#include <stdio.h>

main()
{
        struct SREGS segregs;

        segread(&sregs);
        printf("CS = %04x¥n", sregs.cs);
}
```

atexit

■書式

#include <stdlib.h>

int atexit (func)；

void (＊func) (void)；　　関数へのポインタ

■戻り値

正常に終了した場合は0を返します。エラーが発生した場合は0以外の値を返します。

■機能

正常終了時に関数funcを呼び出すようにします。設定できる関数は最大32個までで、複数の関数を指定する場合には、atexitを複数回呼び出します。複数の指定を行った場合、最後に設定した関数から順に呼び出します。

■ポイント

abort関数や＾Cによる中断では、atexit関数で設定された関数の呼び出しは行いません。

■例

引数で指定された複数のテキストファイルを標準出力に連続して出力します。出力する際に、エスケプシーケンスを用いて、文字色を水色に変更、終了時にはatexit関数で登録された関数が色を元に戻します。

```
#include <stdio.h>
#include <stdlib.h>

/* 色を元に戻す関数 */
void resumecol(void)
{
        fputs("¥033[m", stdout);
}

int main(int argc, char **argv)
{
        FILE *fp;
        int c;

        if (atexit(resumecol) != 0) {
                fputs("atexitで関数が登録できません¥n", stderr);
                return 2;
        }
        fputs("¥033[36m", stdout);
        while (--argc > 0) {
                if ((fp = fopen(*++argv, "r")) == NULL) {
                        perror(*argv);
                        continue;
                }
                while ((c = fgetc(fp)) != EOF)
                        fputc(c, stdout);
                fclose(fp);
        }
        return 0;
}
```

abort　ver 5＃

■書式

＃include ＜process.h＞
＃include ＜stdlib.h＞
(process.hかstdlib.hのいずれか)

void abort (void)；

引数はありません。

■戻り値

ありません。

■機能

以下のメッセージを標準エラー出力stderrに出力した後で、現在のプロセスを終了させます。このときにオープンされているストリームバッファはフラッシュされません。また親のプロセス(あるいはOS)に終了コード3を返します。

```
Abnormal program termination
```

■ポイント

この関数はonexit関数またはatexit関数が実行されていても、それらが指定した関数は起動されません。反対に、signal関数によってSIGABRTが設定されている場合には、その関数が起動されます。ただしSIGABRTへの対応はVer 5からです。

■例

プロセスを異常終了させ、シグナルを検出します。

```
#include        <stdio.h>
#include        <process.h>
#include        <signal.h>
#include        <stdlib.h>


main()
{
        void    on_abort(void);
        void    on_exit(void);

        signal(SIGABRT, on_abort);
        onexit(on_exit);

        abort();        /*      or exit(); */
}

void    on_abort()
{
        printf("¥nThis process is terminated by abort().¥n");
}

void    on_exit()
{
        printf("This process is terminated by exit().¥n");
}
```

cwait　ver 5

■書式

```
#include <process.h>

int cwait (status, pid, action);

int *status;      終了ステータス
int pid;          プロセスID
int action;       動作コード
```

■戻り値

正常に終了した場合、終了したプロセスのプロセスIDを返します。エラーが発生した場合は−1を返し、errnoにはエラー理由を示す以下の定数が設定されます。

記号定数	エラー内容
EINVAL	不正なaction
EINTR	子プロセスが異常終了した。
ECHILD	子プロセスは存在していない。

■機能

pidで指定される子プロセスの終了を待ち、終了した子プロセスのプロセスIDを返します。子プロセスが異常終了した場合や、子プロセスが存在していない場合には−1を返します。statusは終了したプロセスの終了ステータスワードとリターンコードを格納する領域へのポインタです。actionにはprocess.hで定義される以下の定数を与えます。

記号定数	意味
WAIT_CHILD	子プロセスが終了するまで待ちます。
WAIT_GRANDCHILD	子プロセス以下のすべてのプロセスが終了するまで待ちます。

■ポイント

この関数はMS-DOSでは使用できません。MS-OS/2プロテクトモード専用です。

execXXX

[･lpe,･vpe]ver 4

■書式

```
#include <process.h>

int execl    (path, arg0, arg1, ..., argN, NULL);
int execle   (path, arg0, arg1, ..., argN, NULL, envp);
int execlp   (path, arg0, arg1, ..., argN, NULL);
int execlpe  (path, arg0, arg1, ..., argN, NULL, envp);
int execv    (path, argv);
int execve   (path, argv*, envp);
int execvp   (path, argv*);
int execvpe  (path, argv*, envp);

char *path;                          起動するファイル名
char *arg0, *arg1, ..., *argN;       引数のポインタ
char *argv[];                        引数のポインタ配列
char *envp[];                        環境変数のポインタ配列
```

■戻り値

通常は関数から戻ることはありません。ただし、関数の実行に失敗すると−1を返し、errnoに以下の値をセットします。

記号定数	エラー内容
E2BIG	引数並びが128バイトを越えました。
EACCES	指定されたファイルがロックされているかシェアリングが禁止されています。(MS-DOS 3.0以降)
EMFILE	オープンしているファイルの数が多すぎます。
ENOENT	指定したファイルが見つかりません。
ENOEXEC	指定したファイルは実行可能でないか、あるいは内容が破壊されています。

■機能

子プロセスを起動し実行します。この時、子プロセスは親プロセスの存在していたメモリにロードされるので、親プロセスへは帰れなくなります。pathはフルパスでもファイル名だけでもかまいませんが、拡張子については以下の規則が適用されます。

拡張子なし	path, path.COM, path.EXE　の順で検索
拡張子あり	pathのみ

子プロセスに引数や環境を渡すこともできますが、引数を渡す場合には128バイト以内でなければなりません。

関数名の末尾文字はそれぞれ以下の機能を示しています。

文字	機能
p	ファイルを検索する際に環境変数PATHを参照します。
l	子プロセスへの引数をリスト並びで指定します。
v	子プロセスへの引数をポインタ配列で渡します。
e	子プロセスの環境変数をポインタ配列で指定します。

■ポイント

spawnと違って親プロセスがいなくなります。(親プロセスが入れ替わります)

■例

環境変数PROCESS_NAMEとUSERを設定してsymdebコマンドを起動します。

```
#include        <stdio.h>
#include        <process.h>

main()
{
        static char     *env[3]={"PROCESS_NAME=SAMPLE.EXE", "USER=GUEST", NULL};

        execlpe("symdeb", "symdeb", "sample.exe", NULL, env);
}
```

exit
_exit

■書式

#include <process.h>
#include <stdlib.h>
(process.hかstdlib.hのいずれか)

void exit (rc);
void _exit (rc);

int rc;　　　　終了コード

■戻り値

関数からは戻りません。

■機能

現在のプロセスを終了します。exit関数は終了前にストリームバッファをフラッシュし、atexitやonexit関数が指定した関数をコールしてから終了しますが、_exit関数は以上の処理を行いません。

引数として渡す終了コードの下位1バイトは親プロセス(あるいはOS)に子プロセスの終了コードとして返されます。

■ポイント

終了コードは、通常は正常終了の時0、異常終了の時は0以外を返します。また、終了コードはバッチコマンドのERRORLEVELで参照できます。(abort関数の場合は3を返します)

■例

ファイルをオープンし、オープンに失敗した場合はエラーメッセージを表示し、プロセスを終了します。

```
#include        <stdio.h>

main()
{
        FILE    *fp;

        if( (fp=fopen("sample.dat", "rt"))==NULL ) {
                fprintf(stderr, "Can not open 'SAMPLE.DAT'.¥n");
                exit( 1 );
        }

        .
        .
        .
}
```

getpid

■書式

＃include ＜process.h＞

int getpid（void）;

引数はありません。

■戻り値

OSが管理する現在のプロセスIDを返します。

■機能

現在のプロセスIDを通知します。

onexit　　ver 4, ver 5#

■書式

#include <stdlib.h>

onexit_t onexit (func)

onexit_t func；　　関数へのポインタ

■戻り値

正常に終了した場合funcを返します。エラーが発生した場合はNULLを返します。

■機能

正常終了時にユーザーが作成した関数funcを呼び出すようにします。設定できる関数は最大32個までで、複数の関数を指定する場合には、onexitを複数回呼び出します。複数の指令を行った場合、最後に設定した関数から順に呼び出します。引数funcとonexit関数の戻り値の型であるonexit_tは、stdlib.hで次のように定義されています。

```
typedef int (cdecl * cdecl onexit_t) ()；
```

つまり、onexit関数の戻り値・引数funcの型は、intを返す関数へのポインタとなります。

■ポイント

abort関数や^Cによる中断では、onexit関数で設定された関数の呼び出しを行いません。

onexit関数はMicrosoftC独自の関数で、旧バージョンとの互換性のために用意されています。同様の機能がatexit関数で実現できますので、通常はatexit関数を使用してください。atexit関数は、ANSI規格案に準拠しています。

■例

atexit関数での例と同じものをonexit関数を用いて実現したものです。呼び出される関数の型、onexit関数の返すエラー値の違いに注目してください。

```
#include <stdio.h>
#include <stdlib.h>

int resumecol(void)
{
        fputs("¥033[m", stdout);
        return 0;
}

int main(int argc, char **argv)
{
        FILE *fp;
        int c;

        if (onexit(resumecol) == NULL) {
                fputs("onexitで関数が登録できません¥n", stderr);
                return 2;
        }
        fputs("¥033[36m", stdout);
        while (--argc > 0) {
                if ((fp = fopen(*++argv, "r")) == NULL) {
                        perror(*argv);
                        continue;
                }
                while ((c = fgetc(fp)) != EOF)
                        fputc(c, stdout);
                fclose(fp);
        }
        return 0;
}
```

spawnXXX

[･lpe,･vpe]ver 4

■書式

```
#include <process.h>

int spawnl   (mode, path, arg0, arg1, ..., argN, NULL);
int spawnle  (mode, path, arg0, arg1, ..., argN, NULL, envp);
int spawnlp  (mode, path, arg0, arg1, ..., argN, NULL);
int spawnlpe (mode, path, arg0, arg1, ..., argN, NULL, envp);
int spawnv   (mode, path, argv);
int spawnve  (mode, path, argv*, envp);
int spawnvp  (mode, path, argv*);
int spawnvpe (mode, path, argv*, envp);

int  mode;                          親プロセスの実行モード
char *path;                         起動するファイル名
char *arg0, *arg1, ..., *argN;      引数のポインタ
char *argv[];                       引数のポインタ配列
char *envp[];                       環境変数のポインタ配列
```

■戻り値

戻り値は関数の実行が失敗した時は−1を返し、それ以外の時は子プロセスの終了コードを返します。関数の実行に失敗した時はerrnoに以下の値をセットします。

記号定数	エラー内容
E2BIG	引数並びが128バイトを越えました。
ENOENT	指定したファイルが見つかりません。
ENOEXEC	指定したファイルは実行可能でないか、あるいは内容が破壊されています。
EINVAL	modeが不正です。
ENOMEM	子プロセス実行のためのメモリが足りません。

■機能

子プロセスとして、新しいプロセスを起動します。またこの時の親プロセスの実行モードをmodeで指定します。指定可能なモードは、

モード	実行状態
P_WAIT	子プロセスの実行が終了するまで親プロセスの実行は保留されます。
P_OVERLAY	親プロセスを破棄し子プロセスを起動します。(exec関数と同じ結果)

であり、P_OVERLAYを指定した場合にはexec関数と同じ結果になり、基本的にはspawn関数から返ってきません。

pathはフルパスでもファイル名だけでもかまいませんが、拡張子については以下の規則が適用されます。

拡張子なし	path, path.COM, path.EXE	の順で検索
拡張子あり	pathのみ	

子プロセスに引数や環境を渡すこともできますが、引数を渡す場合には128バイト以内でなければなりません。

関数名の末尾文字はそれぞれ以下の機能を示しています。

文字	機能
p	ファイルを検索する際に環境変数PATHを参照します。
l	子プロセスへの引数をリスト並びで指定します。
v	子プロセスへの引数をポインタ配列で渡します。
e	子プロセスの環境変数をポインタ配列で指定します。

■ポイント

P_WAITを指定した場合には親プロセスがメモリ上に存在したままになるので、子プロセスを実行するためのメモリが十分あることを確認してください。

■例

samという名前のコマンドを実行し、エラーレベルを表示します。実行できなかった場合は −1を表示します。

```
#include        <process.h>
#include        <stdio.h>


main()
{
        int             rc;

        rc      = spawnl(P_WAIT, "sam", "sam", NULL);
        printf("RC=%d¥n", rc);
}
```

system　　ver 5#

■書式

#include <**process.h**>
#include <**stdlib.h**>
(process.hかstdlib.hのいずれか)

int system (cmd);

char *cmd;　　コマンド文字列のポインタ

■戻り値

関数が正常終了したときは0，異常終了したときは0以外を返し、errnoに以下の値をセットします。

記号定数	エラー内容
E2BIG	引数並びが128バイトを越えました。
ENOENT	コマンドインタープリタが見つかりません。
ENOEXEC	コマンドインタープリタのファイルが破壊されています。
ENOMEM	コマンドインタープリタを実行するためのメモリが足りません。

■機能

コマンドインタープリタ(COMMAND.COM)を起動し、cmdで指定されたコマンド(DIR,TYPEなど)の処理を依頼します。

■ポイント

コマンドインタープリタを検索するために、環境変数COMSPECおよびPATHが参照されます。

■例

COMMAND.COM を実行し、ディレクトリを表示します。

```
#include        <process.h>
#include        <stdio.h>


main()
{
        int             rc;

        rc      = system("dir /w");
        printf("RC=%d¥n", rc);
}
```

wait　　ver 5

■書式

#include <process.h>

int wait (status);

int *status;　　終了ステータス

■戻り値

正常に終了した場合、終了した子プロセスのプロセスIDを返します。エラーが発生した場合は−1を返し、errnoにはエラー理由を示す以下の定数が設定されます。

記号定数	エラー内容
EINTR	子プロセスが異常終了した。
ECHILD	子プロセスは存在していない。

■機能

子プロセスの終了を待ち、終了したプロセスのプロセスIDを返します。子プロセスが異常終了した場合や、子プロセスが存在していない場合には−1を返します。statusは終了したプロセスの終了ステータスワードとリターンコードを格納する領域へのポインタです。

■ポイント

この関数はMS-DOSでは使用できません。MS-OS/2プロテクトモード専用です。

raise

■書式

```
#include <signal.h>

int raise (sig);

int sig;        シグナル名
```

■戻り値

正常終了時には0、異常終了には0以外を返します。

■機能

sigで指定したシグナルを実行中のプロセスに送ります。sigには以下のものが指定可能です。

シグナル	意味／規定ハンドラの動作
SIGABRT	異常終了しました。プログラムを終了させ、終了コード3を返す。
SIGFPE	浮動小数点の演算においてエラーが発生しました。プログラムを終了させます。
SIGILL	無効な機械語命令がありました。プログラムを終了させます。
SIGINT	CTRL＋Cの割り込みが発生しました。INT 23hを実行します。
SIGSEGV	記憶領域への不正なアクセスがありました。プログラムを終了させます。
SIGTERM	プログラムに終了が要求されました。プログラムを終了させます。

■ポイント

MS-DOSが実際に発生するシグナルはSIGABRT, SIGINT, SIGFPEだけです。また、この関数はMS-DOS Ver3.XXから使用可能です。

■例

^Cキーによる割り込みを発生させます。

```
#include        <conio.h>
#include        <stdio.h>
#include        <signal.h>

int on_ctrl_c(void);

main()
{
        signal(SIGINT, on_ctrl_c);

        raise(SIGINT);
}

int on_ctrl_c()
{
        cprintf("CTRL+C Interrupt.¥n");
}
```

signal

■書式

```
#include <signal.h>

void ( *signal (sig, func) ) ();

int sig;                シグナル名
void (*func) ();        実行される関数
```

■戻り値

関数が正常終了した場合は、signal関数を実行する前のハンドラのアドレスを返し、異常終了した場合には-1を返し、errnoにEINVALがセットされます。

■機能

シグナル割り込みのハンドラを設定します。設定できるシグナルとハンドラは以下の通りです。

シグナル	意味
SIGABRT	異常終了
SIGBREAK STOP	シグナル
SIGFPE	浮動小数点エラー
SIGILL	機械語の無効命令
SIGINT	CTRL+Cシグナル
SIGSEGV	不正なメモリアクセス
SIGTERM	終了要求

ハンドラ	動作
SIG_DFL	プロセスを終了させます。この時、オープンされているストリームバッファはフラッシュされません。
SIG_IGN	シグナルを無視します。(ただし、SIGFPEには設定不可)
func	ユーザーが定義した関数を実行します。

SIGFPEは例外で、SIG_IGN を設定できません。またSIGFPEのハンドラからのリターンはlongjmp関数を使用しなければなりません。

さらに、SIGFPEのハンドラには、第1引数にSIGFPE, 第2引数にFPE_xxx(float.hで定義されている)が渡されます。ユーザーハンドラではFPE_xxxを検査することで対応する処理を行うことができます。

■ポイント

MS-DOSが実際に発生するシグナルはSIGABRT, SIGINT, SIGFPEだけです。また、この関数はMS-DOS Ver3.XX以前のものでは動作が若干異なります。

■例

^Cキーによる割り込みを発生させます。

```
#include        <stdio.h>
#include        <signal.h>
#include        <process.h>


void    on_stop( void );


main()
{
        signal(SIGINT, on_stop);

        raise( SIGINT );

        printf("正常終了¥n");
}

void    on_stop()
{
        printf("ＣＴＲＬ＋Ｃが押されました。¥n");
        exit( 1 );
}
```

chdir

■書式

#include <direct.h>

int chdir (path);

char *path;　　ディレクトリ名

■戻り値

カレントディレクトリの変更に成功した場合は0を返します。失敗した場合は−1を返し、errnoにENOENTをセットします。

■機能

chdir関数はカレントディレクトリをpathに変更します。成功した場合は0を返し、失敗した場合は−1を返してerrnoにENOENTをセットします。

■ポイント

pathにカレントドライブではないドライブを指定した場合、即時的な効果(ドライブが変更される、など)はありません。

■例

カレントディレクトリを変更します

```
#include <stdio.h>
#include <direct.h>

main()
{
        char buf[128];

        puts("ディレクトリ名を入力して下さい");
        putchar(':');
        gets(buf);
        if (*buf == '¥0')
                exit(1);
        if (chdir(buf) != 0)
                puts("そのディレクトリはありません");
        else
                puts("カレントディレクトリを変更しました");
        exit(0);
}
```

getcwd

■書式

#include <direct.h>

char ＊getcwd (path, byte)；

char ＊path；　ディレクトリ名
int byte；　pathの最大長

■戻り値

成功した場合はカレントディレクトリ名の文字列を指すポインタを返します。失敗した場合はNULLを返し、errnoに以下の記号定数をセットします。

記号定数	エラー内容
ENOMEM	pathでNULLが指定されたが、メモリの確保ができなかった。
ERANGE	カレントディレクトリのパス名の文字列がbyteバイトより長い。

これらの記号定数は、errno.hに定義されています。

■機能

getcwd関数はカレントディレクトリのフルパス名を調べ、pathに格納します。その際、長さがbyteで指定した長さより長い場合は、NULLを返してエラーとします。

pathにNULLを指定すると、システムはmalloc関数で領域を確保してからパス名を格納します。

■ポイント

pathにNULLを指定した場合、戻り値のポインタをfree関数に渡して、領域の解放ができます。

■例

カレントディレクトリを表示します

```
#include <stdio.h>
#include <direct.h>

main()
{
        char buf[128];

        if (getcwd(buf, sizeof (buf)) == NULL) {
                printf("カレントディレクトリのパス名が長すぎます。¥n");
                exit(1);
        }
        printf("カレントディレクトリは %s です。¥n", buf);
        exit(0);
}
```

mkdir
rmdir

■書式

#include <direct.h>

int mkdir (path);
int rmdir (path);

char *path;　　ディレクトリ名

■戻り値

成功した場合は0を返します。失敗した場合は−1を返し、errnoに以下の記号定数をセットします。

記号定数	エラー内容
EACCES	ディレクトリが作成できない(mkdir)。
EACCES	以下のうちのいずれかである(rmdir)。 ・ディレクトリが空でない。 ・pathがファイルであった。 ・カレントもしくはルートディレクトリを削除しようとした。
ENOENT	pathが存在しない。

■機能

mkdir関数はpathで指定された名前のディレクトリを作成します。
rmdir関数はpathで指定された名前のディレクトリを消去します。

■ポイント

これらの関数はcommand.comのmd/rdコマンドと似た動作をします。rmdir関数では、ルートディレクトリやカレントディレクトリは削除できません。
pathにNULLを指定した場合、戻り値のポインタをfree関数に渡して、領域の解放ができます。

■例

ディレクトリ¥testdir.tmpを作成し、直後に削除します。

```
#include <stdio.h>
#include <direct.h>

char *tmp = "¥¥testdir.tmp";

main()
{
        if (mkdir(tmp) == -1) {
                perror(tmp);
                exit(1);
        }
        if (rmdir(tmp) == -1) {
                perror(tmp);
                exit(1);
        }
        exit(0);
}
```

memccpy

■書式
#include <memory.h>

void ＊memccpy (dst, src, c, count)；

void ＊dst；　コピー先ポインタ
void ＊src；　コピー元ポインタ
int c；　最終文字
unsigned int count；バイト数

■戻り値
文字cまでコピーされた時は、コピー先中の文字cの位置を指すポインタを返します。
文字cがコピーされなかった時はNULLを返します。

■機能
memccpy関数は、ポインタsrcが指すバッファの内容を、ポインタdstが指すバッファに複写します。memcpyと違い、countバイト分あるいは、文字cが複写されるまで実行されます。

■ポイント
コピー先に十分なメモリが確保されている必要があります。

■例
メモリを確保し、文字列の .(ピリオド)までをコピーします。

```
#include <stdio.h>
#include <memory.h>

main()
{
        char *ptr1, buf2[30];

        ptr1 = "This is a sample. This is not copied";
        memccpy(buf2, ptr1, '.', 30);

        puts(buf2);
}
```

memcpy　　　ver 5#

■書式
#include <memory.h>

void ＊memcpy (dst, src, count)；

void ＊dst；　　　コピー先
const void ＊src；　コピー元
unsigned int count；バイト数

■戻り値
dstのポインタをそのまま返します。

■機能
memcpy関数はポインタsrcの指すバッファの内容を、ポインタdstの指す領域にcountバイト複写します。

■ポイント
memcpy関数では、ふたつの領域に重複している部分があった場合はコピー元の内容を破壊してしまい、結果としてコピー先の内容も破壊されたコピー元の内容が複写されることになります。重複している部分がある場合はmemmove関数を使用します。
memcpy関数はstrcpy関数と違い、'¥0'が現れても複写を続けます。

■例
buf2からbuf1に20バイト転送し、表示します。

```
#include <stdio.h>
#include <memory.h>

main()
{
        char buf1[20], buf2[20];
        int i;
        for (i = 0; i < 20; i++)
                buf2[i] = (char)i;
        buf2[6] = '¥0';
        memcpy(buf1, buf2, 20);
        for (i = 0; i < 20; i++)
                printf("buf1[%2d] ... %d¥t", i, buf1[i]);
}
```

memchr

■書式

#include <memory.h>

void ＊memchr (ptr, c, count)；

const void ＊ptr；	ポインタ
int c；	文字
size_t count；	バイト数

■戻り値

ptrが指すバッファ中に文字cが発見された場合は、発見した位置を指すポインタを返します。発見されなかった場合にはNULLを返します。

■機能

memchr関数はポインタptrが指す領域から、countバイト以内に最初に現れる文字cを探します。検索は文字cが現れるか、countバイトまで調べ終わるまで続けます。

■ポイント

strchr関数では文字列の終了マークである'¥0'が現れると検索を終了してしまいますが、この関数を用いると'¥0'自体やそれ以降も調べることができます。

■例

変数kの内容が0～9までの自乗かどうかチェックします。

```
#include <stdio.h>
#include <memory.h>

main()
{
        char buf[10], *ptr;
        int i, k;

        for (i = 0; i < sizeof buf; i++)
                buf[i] = (char)(i * i);

        k = 64;

        ptr = memchr(buf, k, sizeof buf);

        if (ptr == NULL)
                printf("%dは０～９のいずれの平方でもありません。¥n", k);
        else
                printf("%dは%dの平方です。¥n", k, (int)(ptr - buf));
}
```

memcmp
memicmp

[･icmp]ver 4

■書式

```
#include <memory.h>

int memcmp (buf1, buf2, count);
int memicmp (buf1, buf2, count);

const void *buf1;  バッファ1
const void *buf2;  バッファ2
size_t count;      文字数
```

■戻り値

表の値を返します。

値	意味
< 0	buf1 < buf2
= 0	buf1 = buf2
> 0	buf1 > buf2

■機能

memcmp関数はbuf1とbuf2のふたつのバッファの先頭からcountバイトどうしを比較し、辞書式順序で大小関係を返します。

memicmp関数では、同じアルファベットの大文字と小文字は等しいと見なされます。

■ポイント

STRING関数(strcmpなど)と違って'¥0'以降も比較します。

■例

文字列を比較し、結果を表示します。

```
#include <stdio.h>
#include <memory.h>


main()
{
        char *buf1 = "123abc";
        char *buf2 = "123ABC";
        char *buf3 = "abc123";

        ans(buf1, buf2, 3);
        ans(buf1, buf2, 5);

        ans(buf1, buf3, 3);
}


ans(buf1, buf2, len)
char *buf1, *buf2;
int len;
{
        int rc;
        char fmt[30];

        sprintf(fmt, "%%%d.%dsは%%%d.%dsより大きい¥n", len, len, len, len);

        rc = memcmp(buf1, buf2, len);
        if (rc < 0)
                printf(fmt, buf2, buf1);
        else if (rc == 0)
                printf("文字列どうしは等しい¥n");
        else
                printf(fmt, buf1, buf2);
}
```

memset

■書式

＃include ＜memory.h＞

void ＊memset (ptr, c, count)；

void ＊ptr；　　　　　バッファ
int c；　　　　　　　文字
unsigned int count；バイト数

■戻り値

ptrをそのまま返します。

■機能

memset関数はポインタptrが指す位置からcountバイトを文字cで埋めます。

■ポイント

strnset関数は'¥0'が現れた時点で埋めるのを終了しますが、memset関数では'¥0'以後もcountバイトまで文字cを埋め込みます。

■例

bufを'＊'で初期化します。

```
#include <stdio.h>
#include <memory.h>

main()
{
        char buf[10];
        int i;

        memset(buf, '*', sizeof buf);
        for(i = 0; i < sizeof buf; i++)
                putchar(buf[i]);
}
```

movedata

■書式

#include <memory.h>

void movedata (seg1, ofs1, seg2, ofs2, nbytes)；

unsigned int seg1；	セグメント1
unsigned int ofs1；	オフセット1
unsigned int seg2；	セグメント2
unsigned int ofs2；	オフセット2
unsigned int n；	バイト数

■戻り値

ありません。

■機能

movedata関数はseg1：ofs1によって指定したアドレスからnバイトを、seg2：ofs2によって指定したアドレスにnバイトだけ複写します。

■ポイント

small, mediumモデルの標準メモリ関数では、異なるデータセグメント間のデータの移動を行うことはできません。

この関数はセグメント、オフセットを直接指定してデータを移動させる場合に用います。引数を誤ると致命的な結果となることがありますので、使用には十分注意してください。

memmove　ver 5

■書式

#include <string.h>

void ＊memmove (dst, src, n)

void ＊dst；	コピー先ポインタ
void ＊src；	コピー元ポインタ
unsinged int n；	バイト数

■戻り値

dstを返します。

■機能

srcからdstにnバイトコピーします。領域が重複する場合でも正常にコピーされます。

■ポイント

領域が重複する場合、memcpy関数は正常にコピーを行うことができません。そのような場合に、このmemmove関数を使用してください。重複した領域も正しくコピーされること以外はmemcpy関数と同等です。

■例

重複した領域のコピーを行い、正常にコピーされていることを確認します。

```
#include <string.h>
#include <stdio.h>

char buf[10] = {
        1, 2, 3, 4, 5, 6, 7, 0, 0, 0
};

void main(void)
{
        int i;

        memmove(buf + 3, buf, 7);
        for (i = 0; i < 10; i++)
                printf("%d  ", buf[i]);
        printf("¥n");
}
```

alloca　ver 4

■書式

#include <**malloc.h**>

void ＊alloca (size)

unsigned int size ;　　メモリブロックサイズ

■戻り値

割り当てられたメモリブロックへのポインタを返します。エラーが発生した場合はNULLを返します。

■機能

スタック上にメモリを割り当てます。

■ポイント

malloc関数ではfree関数を用いて解放作業をしなければなりませんが、alloca関数はalloca関数を呼び出した関数が終了すると、自動的に割り当てられた領域を解放します。alloca関数で割り当てた領域をfree関数によって解放することはできません。

スタック上に領域を取るため、実行ファイルのスタックサイズを十分に割り当てておかなければなりません。デフォルトではスタックサイズは2048バイトに設定されています。大きな領域を割り当てたい場合には、clコマンドの/Fオプション、linkコマンドの/stackオプション、exemodコマンドの/stackオプションにより、スタックサイズを拡張してください。

■例

第1引数を小文字に変換し表示します。

```
#include <stdio.h>
#include <malloc.h>
#include <stdio.h>

int main(int argc, char **argv)
{
        char *p;
        int l;

        if (argc != 2) {
                exit(2);
        }
        l = strlen(argv[1]);
        if ((p = alloca(l)) == NULL) {
                perror("alloca error");
                exit(2);
        }
        strcpy(p, argv[1]);
        strlwr(p);
        puts(p);

        exit(0);
}
```

_expand　ver 4

■書式

#include <malloc.h>

void ＊_expand (block, size)

void ＊block；　メモリブロックへのポインタ
unsigned int size；　メモリブロックサイズ

■戻り値

変更されたメモリブロックへのポインタを返します。エラーが発生した場合はNULLを返します。

■機能

blockで示されるメモリブロックのサイズsizeを変更します。realloc関数と違い、メモリブロックの先頭アドレスは変更されることはありません。

■ポイント

領域を拡張する場合、割り当てられている領域の直後に空き領域がないとエラーになります。realloc関数の方が標準的ですので、メモリブロックの先頭アドレスを変更してもよい場合には、realloc関数の使用をお薦めします。

■例

100バイトの領域をふたつ割り当て、両方を200バイトに拡張しようと試みます。この例では、ひとつ目の領域が拡張できずにエラーとなるでしょう。

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

int main(void)
{
        char *p1, *p2;

        if ((p1 = malloc(100)) == NULL) {
                perror("p1");
                return 2;
        }
        if ((p2 = malloc(100)) == NULL) {
                perror("p2");
                return 2;
        }
        if (_expand(p1, 200) == NULL)
                fputs("Cannot expand p1¥n", stderr);
        if (_expand(p2, 200) == NULL)
                fputs("Cannot expand p2¥n", stderr);
        free(p1);
        free(p2);
}
```

free
_ffree
_nfree

[_ffree,_nfree]ver 4

■書式

```
#include <malloc.h>

void free (block);
void _ffree (fblock);
void _nfree (nblock);

void *block;         解放するメモリブロックへのポインタ
void far *fblock;    farヒープ内の解放するメモリブロックへのポインタ
void far *nblock;    nearヒープ内の解放するメモリブロックへのポインタ
```

■戻り値

ありません。

■機能

領域を解放します。

free関数はmalloc, calloc関数等によって割り当てられた領域を解放します。

_ffree関数は_fmalloc関数によって割り当てられたfar領域を解放します。

_nfree関数は_nmalloc関数によって割り当てられたnear領域を解放します。

■ポイント

_ffree, _nfree関数は標準的なライブラリ関数ではないため、ソースレベルでの移植性が低下します。特に必要としない場合にはfree関数のみを使用すべきです。

■例

malloc関数を参照してください。

_heapchk
_nheapchk
_fheapchk

ver 5

■書式

```
#include <malloc.h>

int _heapchk ()
int _nheapchk ()
int _fheapchk ()
```

■戻り値

ヒープ領域の整合性のチェック結果を以下の定数で返します。これらの定数はmalloc.hで定義されています。

記号定数	内容
_HEAPOK	整合性があります。
_HEAPEMPTY	初期化されていません。
_HEAPBADBEGIN	ヘッダ情報が異常です。
_HEAPBADNODE	ノード情報が異常です。

■機能

ヒープ領域の整合性をチェックします。

_heapchkはマクロにより定義され、メモリモデルによって、_nheapchk関数、_fheapchk関数に振り分けられています。

_nheapchk関数はnearヒープを、_fheapchk関数はfarヒープの領域の整合性をチェックします。

■ポイント

ヒープメモリに問題があるプログラムのデバッグに用います。mallocなどの関数で割り当てられたメモリのコントロールブロックが破壊されていないかどうかのチェックを行うことができます。

■例

メモリ割り当てを行い、ヒープの整合性チェックをしています。

```
#include <stdio.h>
#include <malloc.h>
#include <memory.h>

void herror(char *msg, int hcond)
{
        fputs(msg, stderr);
        switch (hcond) {
        case _HEAPOK:
                fputs("整合性があります¥n", stderr);
                break;
        case _HEAPEMPTY:
                fputs("初期化されていません¥n", stderr);
                break;
        case _HEAPBADBEGIN:
                fputs("ヘッダ情報が異常です¥n", stderr);
                break;
        case _HEAPBADNODE:
                fputs("ノード情報が異常です¥n", stderr);
                break;
        default:
                fputs("unknown¥n", stderr);
        }
}

void main(void)
{
        char *pp[10];
        int i;

        herror("最初          : ", _heapchk());
        for (i = 0; i < 10; i++)
                pp[i] = malloc(20);
        herror("割り当て後    : ", _heapchk());
}
```

heapset
_nheapset
_fheapset

ver 5

■書式

#include <malloc.h>

int _heapset (c)
int _nheapset (c)
int _fheapset (c)

unsigned int c；　　充塡する文字

■戻り値

ヒープ領域の整合性のチェック結果を以下の定数で返します。これらの定数はmalloc.hで定義されています。

記号定数	内容
_HEAPOK	整合性があります。
_HEAPEMPTY	初期化されていません。
_HEAPBADBEGIN	ヘッダ情報が異常です。
_HEAPBADNODE	ノード情報が異常です。

■機能

ヒープ領域の整合性をチェックし、ヒープのフリー領域にcを書き込みます。

_heapsetはマクロにより定義され、メモリモデルによって、_nheapset関数、_fheapset関数に振り分けられています。

_nheapset関数はnearヒープを、_fheapset関数はfarヒープの領域の整合性をチェックし、cを書き込みます。

■ポイント

ヒープメモリに問題のあるプログラムのデバッグに用います。

■例

割り当て後に_heapsetを行った結果、解放後に_heapsetを行った結果を表示します。

```
#include <stdio.h>
#include <malloc.h>
#include <memory.h>

void herror(char *msg, int hcond)
{
        fputs(msg, stderr);
        switch (hcond) {
        case _HEAPOK:
                fputs("整合性があります¥n", stderr);
                break;
        case _HEAPEMPTY:
                fputs("初期化されていません¥n", stderr);
                break;
        case _HEAPBADBEGIN:
                fputs("ヘッダ情報が異常です¥n", stderr);
                break;
        case _HEAPBADNODE:
                fputs("ノード情報が異常です¥n", stderr);
                break;
        default:
                fputs("unknown¥n", stderr);
        }
}

void dump(void *p, int n)
{
        unsigned int *q;

        q = p;
        while (n-- > 0)
                printf("%04x ", *q++);
        putchar('¥n');
}


void main(void)
{
        char *p;

        p = malloc(16);
        memset(p, '¥0', 16);
        dump(p, 8);
        herror("割り当て後   : ", _heapset(0xaaaa));
        dump(p, 8);
        free(p);
        herror("解放後       : ", _heapset(0x7070));
        dump(p, 8);
}
```

_heapwalk
_nheapwalk
_fheapwalk

ver 5

■書式

#include ＜malloc.h＞

int _heapwalk (entry)
int _nheapwalk (fpentry)
int _fheapwalk (npentry)

struct _heapinfo *entry；	構造体へのポインタ
struct _heapinfo far *fentry；	構造体へのポインタ
struct _heapinfo near *nentry；	構造体へのポインタ

■戻り値

ヒープ領域の整合性のチェック結果を以下の定数で返します。これらの定数はmalloc.hで定義されています。

記号定数	内容
_HEAPOK	整合性があります。
_HEAPEMPTY	初期化されていません。
_HEAPBADBEGIN	ヘッダ情報が異常です。
_HEAPBADNODE	ノード情報が異常です。
_HEAPEND	ヒープの終わりとなった。

■機能

ヒープエントリの情報を取得します。引数entryは構造体_heapinfoへのポインタで、構造体_heapinfoはmalloc.hで次のように定義されています。

```
typedef struct _heapinfo  {
        int far * _pentry;
        size_t _size;
        int _useflag;
}          _HEAPINFO;
```

_heapwalkは、_heapinfo構造体のメンバ_pentryで示されるヒープ領域の次の領域の情報をentryに返します。_pentryにNULLを代入して_heapwalk関数を呼び出すと、最初のヒープエントリの情報が取得できます。連続的に呼び出すことにより、全ヒープ領域の情報を取得することができます。

取得すべきエントリの終わりに達すると、_heapwalk関数は定数_HEAPENDを返します。

_heapwalkはマクロにより定義され、メモリモデルによって、_nheapwalk関数、_fheapwalk関数に振り分けられています。

_nheapwalk関数はnearヒープを、_fheapwalk関数はfarヒープの領域の整合性をチェックし各エントリの情報を取得します。

■ポイント

ヒープメモリに問題のあるプログラムのデバッグに用います。

■例

割り当て・解放を行った後のヒープの状態を表示します。

```
#include <stdio.h>
#include <malloc.h>
#include <dos.h>

void main(void)
{
        char *pp[4];
        struct _heapinfo entry;

        pp[0] = malloc(3);
        pp[1] = malloc(10);
        pp[2] = malloc(150);
        pp[3] = malloc(20);
        free(pp[1]);
        pp[1] = malloc(40);

        entry._pentry = NULL;
        while (_heapwalk(&entry) != _HEAPEND ) {
                printf("seg:off = %04x:%04x %d bytes ... %s¥n",
                        FP_SEG(entry._pentry),
                        FP_OFF(entry._pentry),
                        entry._size,
                        entry._useflag ? "used" : "free"
                );
        }
}
```

malloc
_fmalloc
_nmalloc

[malloc]ver 5#
[_f・, _n・]ver 4

■書式

#include <malloc.h>

void ＊malloc (size)；
void far ＊fmalloc (size)；
void near ＊nmalloc (size)；

unsigned int size；　　メモリブロックサイズ

■戻り値

割り当てられたメモリブロックへのポインタを返します。エラーが発生した場合はNULLを返します。

■機能

sizeバイトの領域を割り当てます。
_fmalloc関数はfarヒープから領域を割り当てます。
_nmalloc関数はnearヒープから領域を割り当てます。

■ポイント

_fmalloc, _nmalloc関数は標準的なライブラリ関数ではないため、ソースレベルでの移植性が低下します。特に必要でない場合にはmalloc関数のみを使用すべきです。

■例

メモリアロケーションを行い、データを書き込んだ後に解放します。

```
#include <stdio.h>
#include <malloc.h>
#include <memory.h>

void main(void)
{
        char *pp[100];
        int i;

        for (i = 0; i < 100; i++)
                pp[i] = malloc(i);

        for (i = 0; i < 100; i++)
                memset(pp[i], i, i);

        for (i = 0; i < 100; i++)
                free(pp[i]);
}
```

_msize
_fmsize
_nmsize

ver 4
[_msize]ver 5#

■書式

#include <malloc.h>

unsigned int _msize (block);
unsigned int _fmsize (fblock);
unsigned int _nmsize (nblock);

void *block;	メモリブロックを指すポインタ
void far *fblock;	メモリブロックを指すポインタ
void near *nblock;	メモリブロックを指すポインタ

■戻り値

エラー戻り値はありません。

■機能

_msize関数はmalloc関数等で割り当てられた領域blockの大きさを返します。
_fmsize関数は_fmalloc関数等で割り当てられたfar領域fblockの大きさを返します。
_nmsize関数は_nmalloc関数等で割り当てられたnear領域nblockの大きさを返します。

■ポイント

これらの関数は標準的なライブラリ関数ではないため、使用すると移植性が低下します。割り当てた領域のサイズはプログラマが管理すべきで、特に必要のない限りこれらの関数は使用すべきではありません。

■例

メモリを割り当て、割り当てられたブロックのサイズを表示します。

```
#include <stdio.h>
#include <malloc.h>

void main(void)
{
        void *p1, *p2, *p3;

        p1 = malloc(1000);
        p2 = malloc(2000);
        p3 = malloc(9999);

        printf("size of block p1 = %u¥n", _msize(p1));
        printf("size of block p2 = %u¥n", _msize(p2));
        printf("size of block p3 = %u¥n", _msize(p3));

        free(p1);
        free(p2);
        free(p3);
}
```

_freect　ver 4

■書式

＃include ＜malloc.h＞

unsigned int _freect (size)

unsigned int size；　　データ項目ひとつあたりのバイト数

■戻り値

エラー戻り値はありません。

■機能

sizeバイトのmalloc関数によるメモリ割り当てを、何回繰り返すことができるかを返します。

■ポイント

返される値は正確なものではなく、多少の誤差を含みます。

■例

10バイトのメモリ割り当てが何回繰り返せるか表示します。

```
#include <stdio.h>
#include <malloc.h>

void main(void)
{
        printf("%d¥n", _freect(10));
}
```

halloc　　ver 4

■書式

#include 〈**malloc.h**〉

void huge ＊halloc (n, size)；

long n；　　配列要素の数
unsigned int size；　　１要素のバイト数

■戻り値

割り当てられた領域へのhugeポインタを返します。エラーが発生した場合にはNULLを返します。

■機能

1要素がsizeバイト、n要素からなるhuge配列を割り当てます。

■ポイント

64キロバイトを越える領域を割り当てる際に用います。

割り当てる大きさが128キロバイトを越える場合には、sizeは2のべき乗(1,2,4,8,16...)でなければなりません。

■例

5万要素のlong配列をhallocにより割り当て、数値を代入、解放します。

```
#include <stddef.h>
#include <malloc.h>
#include <stdlib.h>

int main(void)
{
        long l;
        long huge *hp, huge *hp0;

        if ((hp0 = (long huge *)halloc(50000L, sizeof(long))) == NULL) {
                perror("halloc");
                return 2;
        }
        hp = hp0;
        for (l = 0; l < 50000L; l++)
                *hp++ = l;
        hfree(hp0);
        return 0;
}
```

hfree　　ver 4

■書式

```
#include <malloc.h>

void hfree (hblock);
void huge *hblock;
```

■戻り値

ありません。

■機能

halloc関数によって割り当てられた領域を解放します。

■例

halloc関数を参照してください。

_memavl　　ver 4

■書式

＃include ＜malloc.h＞

unsigned int _memavl (void);

■戻り値

エラー戻り値はありません。

■機能

ヒープ領域の空き容量をバイト単位で返します。

■ポイント

返される値は正確なものではなく、多少の誤差を含みます。

■例

メモリ割り当て・解放に従って変化する残りメモリ容量を表示します。

```
#include <stdio.h>
#include <malloc.h>

void printav(void)
{
        printf("%6u bytes free¥n", _memavl());
}

void main(void)
{
        char *p1, *p2;

        printav();
        p1 = malloc(1000);
        printav();
        p2 = malloc(500);
        printav();
        free(p1);
        printav();
        free(p2);
        printav();
}
```

_memmax　ver 5

■書式

#include <malloc.h>

unsigned int _memmax (void)；

■戻り値

エラー戻り値はありません。

■機能

nearヒープの割り当て可能な最大ブロックサイズを返します。

■例

メモリ割り当て・解放に従って変化する最大ブロックサイズを表示します。

```
#include <stdio.h>
#include <malloc.h>

void printav(void)
{
        printf("%6u bytes free¥n", _memmax());
}

void main(void)
{
        char *p1, *p2;

        printav();
        p1 = malloc(1000);
        printav();
        p2 = malloc(500);
        printav();
        free(p1);
        printav();
        free(p2);
        printav();
}
```

calloc

■書式

#include <malloc.h>

void ＊calloc (count, size)

unsigned int count；　　配列要素の数
unsigned int size；　　1要素あたりのバイト数

■戻り値

割り当てられたメモリブロックへのポインタを返します。エラーが発生した場合はNULLを返します。

■機能

各要素のサイズがsizeバイト、要素数がcountの領域を割り当て、0で初期化します。

■ポイント

割り当てられる領域はsize×countバイトとなります。配列領域の割り当てに適しています。

■例

整数の配列を割り当て、初期化、表示します。

```
#include <stdio.h>
#include <malloc.h>

int main(int argc, char **argv)
{
        int *ip;
        int i;

        if ((ip = calloc(10, sizeof(int))) == NULL) {
                perror("calloc error");
                exit(2);
        }
        for (i = 0; i < 10; i++)
                ip[i] = i;
        for (i = 0; i < 10; i++)
                printf("%d¥n", i);
        exit(0);
}
```

realloc

■書式

#include <malloc.h>

void *realloc (block, newsize);

void *block;	メモリブロックへのポインタ
unsigned int newsize;	メモリブロックのサイズ

■戻り値

再割り当てされたメモリブロックへのポインタを返します。エラーが発生した場合はNULLを返します。

■機能

malloc関数等で割り当てられた領域blockのサイズを、newsizeバイトに変更します。

■ポイント

_expand関数と異なり、領域の拡張ができなかった場合は、新たに領域を割り当て、メモリの内容を転送します。

■例

_expand関数のサンプルと同様に、ふたつの100バイトの領域を200バイトずつに拡張します。_expand関数ではエラーとなりますが、この例では双方の領域とも正常に再割り当てされます。

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

int main(void)
{
        char *p1, *p2;

        if ((p1 = malloc(100)) == NULL) {
                perror("p1");
                return 2;
        }
        if ((p2 = malloc(100)) == NULL) {
                perror("p2");
                return 2;
        }
        if ((p1 = realloc(p1, 200)) == NULL)
                fputs("Cannot reallocate p1¥n", stderr);
        else
                fputs("p1 reallocated.¥n", stderr);
        if ((p2 = realloc(p2, 200)) == NULL)
                fputs("Cannot reallocate p2¥n", stderr);
        else
                fputs("p2 reallocated.¥n", stderr);
        free(p1);
        free(p2);
}
```

sbrk

■書式

#include <malloc.h>

void *sbrk (size);

int size;　　　増減するバイト数

■戻り値

変更前のブレークアドレスを返します。エラーが発生した場合は(char *)(-1)を返します。

■機能

プロセスで使用するメモリの最終アドレスをsizeバイト増減させます。

stackavail　ver 4

■書式

#include <malloc.h>

unsigned int stackavail (void);

■戻り値

エラー戻り値はありません。

■機能

スタックから割り当てることのできるメモリのサイズを取得します。

■ポイント

返される値は正確なものではなく、多少の誤差を含みます。

■例

スタックの残りが400バイトを切るまで、再帰的にrec関数を呼び出します。

```
#include <stdio.h>
#include <malloc.h>

void rec(int level)
{
        int sav;

        sav = stackavail();
        printf("%d(%u)", level, sav);
        if (sav > 399) {
                printf(", ");
                rec(level + 1);
        }
        return;
}

void main(void)
{
        rec(1);
        putchar('¥n');
}
```

atoi
atol

■**書式**

#include <stdlib.h>

int atoi (str)；
long atol (str)；

const char ＊str；　　　　数値へ変換する文字列

■**戻り値**

変換した結果をintまたはlongで返します。変換不可能な場合は、0(atolでは0L)を返します。

■**機能**

atoi関数は文字列strを10進数の文字列と見なし、int型の数値に変換します。

atof関数は文字列strを10進数の文字列と見なし、long型の数値に変換します。

どちらの関数も、数値の一部として認識できない文字が現れる時点までを数値として変換します。たとえば"128KR1"という文字列では、数値は128になります。

■**ポイント**

与える文字列strの最初の部分には、空白文字(whitespace)があってもかまいません。また、+/－も認識します。

■**例**

引数を数値に変換した後に表示します。int型の範囲を越えた数値を指定すると、atolとatoiの差異がわかりやすいでしょう。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        long l;
        int i;

        if (argc != 2) {
                fputs("引数を1つ付けてください¥n", stderr);
                return 1;
        }
        i = atoi(argv[1]);
        l = atol(argv[1]);
        printf("i = %d¥n", i);
        printf("l = %ld¥n", l);
        return 0;
}
```

atof

■書式

#include <math.h>
#include <stdlib.h>
(math.hかstdlib.hかいずれか)

double atof (string);

const char *string;　　変換対象文字列

■戻り値

変換対象文字列が変換された倍精度浮動小数点数値を返します。文字列が変換不能ならば戻り値は0となります。オーバーフローが発生した場合、戻り値は不定となります。

■機能

文字列を倍精度浮動小数点数へ変換します。被変換文字列は以下の形式である必要があります。

```
<whitespase> < {+|-}> <digits> <.digits> < {d|D|e|E}  <sign> digits>
                       整数部     小数部
         |---------- 仮数部 --------| |------- 指数部 ---------|
```

whitespace : tab / space
sign　　　 : + / −
digits　　 : 10進数

なお、仮数整数部が無指定の場合は仮数小数部に最低限ひとつの数字を指定しなければなりません。指数部はDまたはEで始まり、その後に符号付き10進数が続きます。

■ポイント

atof関数は認識できない文字(ASCIZの最後の¥0等)を見つけた時点で数値変換を中止します。

データ変換関数

itoa
ltoa
ultoa

■書式

#include <stdlib.h>

char * itoa (int_num, buf, base);
char * ltoa (long_num, buf, base);
char * ultoa (ulong_num, buf, base);

int int_num;	変換するint型数値
long long_num;	変換するlong型数値
unsigned long ulong_num;	変換するunsigned long型数値
char * buf;	変換結果の格納先
int base;	変換する基数

■戻り値

変換結果の文字列を指すポインタ(==buf)を返します。エラー戻り値はありません。

■機能

itoa,ltoa,ultoa関数は、指定された数値(int_num,long_num,ulong_num)をbase進数の文字列に変換します。itoa関数はint型、ltoa関数はlong型、ultoa関数はunsigned long型の数値を文字列に変換します。

itoa,ltoa関数では、変換された結果に'−'符号が付くのはbaseに10を指定した場合のみです。

■ポイント

baseに11以上の数値を指定した場合は、A～Zが10～35に割り当てられます。

■例

数値の基数変換をします。元の数値の基数、変換先の基数、数値を引数にとり起動すると変換結果を表示します。

```
#include <stdio.h>
#include <stdlib.h>

void usage(char *name)
{
        fprintf(stderr, "usage: %s src-base dst-base number ...¥n", name);
        exit(1);
}

int main(int argc, char **argv)
{
        int base1, base2;
        long l;
        char *p, outbuf[32];

        if (--argc <= 0)
                usage(argv[0]);
```

```
	if ((base1 = atoi(*++argv)) == 0)
		usage(argv[0]);
	if (--argc <= 0)
		usage(argv[0]);
	if ((base2 = atoi(*++argv)) == 0)
		usage(argv[0]);
	while (--argc > 0) {
		l = strtol(*++argv, &p, base1);
		*p = '¥0';
		ltoa(l, outbuf, base2);
		printf("%s(%d) == %s(%d)¥n", *argv, base1, outbuf, base2);
	}
	return 0;
}
```

ecvt
fcvt

■書式

#include <stdlib.h>

char *ecvt (num, count, dec, sign);
char *fcvt (num, count, dec, sign);

double num; 変換する数値
int count; 有効桁数(ecvt)/小数点以下の桁数(fcvt)
int *dec; 小数点位置
int *sign; 符号

■戻り値

変換した結果の数値部分を返し、decに小数点位置、signに符号をセットします。エラー戻り値はありません。

■機能

ecvtとfcvt関数は、double型の数値numを文字列に変換します。numに変換したい数値を、countに変換するときの有効桁数count(fcvt関数では小数点以下の桁数)を与えて、ecvtやfcvt関数を実行します。ecvtやfcvt関数は結果の数値部分のみ(小数点記号や符号は含まれない)を戻り値として返し、decに小数点の位置、signに符号をセットして戻ります。

decに返される数値は戻り値の文字列の先頭から数えた小数点の位置です。たとえば、戻り値のポインタが"314159"を指していて、decに1がセットされた場合は"3.14159"となります。decに0以下の数値がセットされた場合は、小数点は文字列の最初に位置(例:".02"など)になります。

signに返される数値は符号を表わしています。符号が正(+)であれば0を、負(−)であれば0以外がセットされます。

■ポイント

numの桁数がcountを越えていた場合はcount桁まで丸めます。短かすぎる場合は'0'で埋めます。 decとsignはint型へのポインタ(int num;として&numなど)を与えるようにします。

ecvt関数とfcvt関数の違いは、countに与える数値がecvt関数の場合は有効桁数、fcvt関数の場合は小数点以下の有効桁数であることです。ecvt関数とfcvt関数は同じバッファを使用しますので、ecvtやfcvt関数を2度実行した場合は、以前の変換結果は残りません。

■例

引数で指定された数値の平方根を表示します。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char **argv)
{
        double x;
        char *endptr, *p;
        int point, sign;

        if (argc != 2) {
                fputs("平方根を求めます。¥n", stderr);
                return 2;
        }
        x = sqrt(strtod(argv[1], &endptr));
        p = fcvt(x, 10, &point, &sign);
        while (point-- > 0)
                putchar(*p++);
        putchar('.');
        while (*p != '¥0')
                putchar(*p++);
        putchar('¥n');
        return 0;
}
```

gcvt

■書式
#include <stdlib.h>

char *gcvt (num, count, buffer);

double num; 変換する数値
int count; 有効桁数
char *buffer; 結果の格納先

■戻り値
変換した結果、つまりbufferを返します。エラーを示す戻り値はありません。

■機能
gcvt関数はdouble型の数値を直接文字列に変換し、bufferに格納します。countには有効桁数を指定します。有効桁数内に数値が納まらない場合は指数形式になります。

■ポイント
ecvtやfcvt関数と違い、直接符号や小数点、指数などを含む文字列に変換します。bufferには文字列を格納できる十分なメモリがなくてはなりません。

■例

引数で与えられた数値の自然対数を計算します。

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char **argv)
{
        char *p, buffer[16];

        if (argc != 2) {
                fputs("自然対数を計算します¥n", stderr);
                return 2;
        }
        gcvt(log(strtod(argv[1], &p)), 14, buffer);
        puts(buffer);
        return 0;
}
```

strtod
strtol
strtoul

[·tod,·tol]ver 4
[·toul]ver 5

■書式

#include <stdlib.h>

double strtod (ptr, endptr);
long strtol (ptr, endptr, base);
unsigned long strtoul (ptr, endptr, base);

const char *ptr;	変換する文字列
char **endptr;	変換を中止した位置
int base;	文字列中の数値の基数

■戻り値

それぞれ、変換に成功した場合は変換した数値を返し、endptrに変換を中止した文字列の位置をセットします。

変換に失敗した場合はendptrにNULLをセットし、errnoにERANGEをセットして、以下の数値を返します。

現象	関数	戻り値
オーバーフロー	strtod strtol strtoul	±HUGE_VAL LONG_MAXもしくはLONG_MIN ULONG_MAX
アンダーフロー	strtod	0
変換に失敗した	strtod strtol strtoul	0.0 0L 0L

■機能

strtod関数は、文字列ptrを10進浮動小数点数の文字列と見なしてdouble型に変換します。strに与える文字列は、以下の書式でなければなりません。

【+ |−】【0〜9の並び】【.0〜9の並び】【d|D|e|E 【+ |−】 0〜9の並び】】

strtod関数は上記の書式に反する文字が現れると、その時点で変換を終了します。

strtol, strtoul関数は文字列ptrをbase進数の文字列と見なして、long, unsigned long型に変換します。strに与える文字列は、以下の書式でなければなりません。

【+ |−】【0】【x|X】【0〜9 | A〜Zの並び】

+|−はstrtoulでは指定できません。strtol, strtoul関数は、上記の書式に反する文字やbase進数を構成する文字として不適当な文字が現れた時点で変換を終了します。A〜Zには10〜35までの数値が割り当てられています。

BASEに0を指定した場合、strtol, strtoul関数は文字列数値部分の先頭の"0"もしくは"0x","0X"で基数を決定します。'1'〜'9'の数値で始まっている場合は10進数、'0'で始まっている場合は8進数、"0x"の場合は16進数と見なされます。

それぞれの関数とも、変換を中止した文字列の位置をendptrにセットします。

■ポイント

文字列の先頭には空白文字(whitespace)がいくらあってもかまいません。コンパクト、ラージモデルのプログラムの場合、strtod関数の文字列strは100文字以内にします。

■例

このサンプルは基数の変換を行います。base 16 fff と入力すると、16進数fffを変換し、16進・10進・8進にして出力します。base 2 1011100 とすると、2進数1011100について変換します。

```
#include <stdio.h>
#include <stdlib.h>

void usage(char *name)
{
        fprintf(stderr, "usage: %s base number¥n", name);
        exit(1);
}

int main(int argc, char **argv)
{
        int base;
        long l;
        char *p;

        if (--argc <= 0)
                usage(argv[0]);
        if ((base = atoi(argv[1])) == 0)
                usage(argv[0]);
        ++argv;
        while (--argc > 0) {
                ++argv;
                printf("%s(%d) ", *argv, base);
                l = strtol(*argv, &p, base);
                if (*p != '¥0') {
                        printf(" ... error¥n");
                        continue;
                }
                printf("0x%lx, %ld, 0%lo¥n", l, l, l);
        }
        return 0;
}
```

isalnum
isalpha
isascii

■書式

#include <ctype.h>

int isalnum (c);
int isalpha (c);
int isascii (c);

int c;　　　　　文字

■戻り値

それぞれ、条件にあてはまる場合は0以外を、そうではない場合は0を返します。

■機能

isalnumマクロは文字cが英字もしくは数字であるかを調べます。
isalphaマクロは文字cが英字であるかを調べます。
isasciiマクロはASCII文字セット(0x00～0x7f)であるかを調べます。

■ポイント

これらはすべてマクロで定義されています。

■例

入力された文字列中のアルファベット・数字のみを出力します。

```
#include        <stdio.h>
#include        <ctype.h>

main()
{
        int c;

        while((c = getchar()) != EOF) {
                if(isalnum(c))
                        putchar(c);
        }
}
```

iscntrl islower isspace isdigit isprint isupper isgraph ispunct isxdigit

■書式

```
#include <ctype.h>

int iscntrl (c);
int isdigit (c);
int isgraph(c);
int islower (c);
int isprint (c);
int ispunct (c);
int isspace (c);
int isupper (c);
int isxdigit(c);

int c;          文字
```

■戻り値

それぞれ、条件にあてはまる場合は0以外を、そうではない場合は0を返します。

■機能

それぞれ、以下の条件にあてはまるかどうかを調べます。

マクロ名	条件
iscntl	制御文字(0x1f以下および0x7f)
isspace	ホワイトスペース(0x09～0x0d,0x20)
isgraph	表示可能(0x21～0x7e)
isprint	表示可能(0x20～0x7e)
isdigit	数字
isxdigit	16進数を構成する文字
islower	英小文字
isupper	英大文字
ispunct	句読点文字

■ポイント

これらのマクロは文字セットとしてASCII文字セット(0x00～0x7f)までを想定しています。文字cにASCII文字セット以外を指定するとEOFを返します。

toascii　tolower　toupper　_tolower　_toupper

■書式

```
#include <ctype.h>

int toascii (c);
int tolower(c);
int _tolower(c);
int toupper(c);
int _toupper(c);

int c;          文字
```

■戻り値

それぞれ、文字cを規則に従って変換した値を返します。

■機能

toasciiマクロは文字cをASCII文字セット(0x00〜0x7f)に変換します。
tolowerマクロは文字cが英大文字の場合、英小文字に変換します。
toupperマクロは文字cが英小文字の場合、英大文字に変換します。
_tolowerマクロは文字cが英大文字であると見なし、英小文字に変換します。
_toupperマクロは文字cが英小文字であると見なし、英大文字に変換します。

■ポイント

_tolowerと_toupperは、文字cがどのような文字であっても変換してしまいます。

iscsymf
iscsym

■書式

```
#include <ctype.h>

int iscsymf(c);
int iscsym (c);

int c;          文字
```

■戻り値

それぞれ、条件にあてはまる場合は0以外を、そうではない場合は0を返します。

■機能

iscsymfマクロはisalphaまたは_であるかを調べます。

iscsymマクロはisalnumまたは_であるかを調べます。

■ポイント

実際には、iscsymfは文字cがシンボル名の第1文字として適当かどうか、iscsymは同じく第2文字以降の文字として適当かを調べる用途に用います。

シンボル名は、_もしくはアルファベットで始まり、_もしくはアルファベット、数字で構成される文字列を指します。この文字列は、Cの変数名などのシンボル名として有効です。

strcat
strncat

■書式

#include <string.h>

char *strcat (str1, str2)；
char *strncat (str1, str2, count)；

char *str1；	文字列1
const char *str2；	文字列2
unsigned int count；	文字数

■戻り値

str1を返します。

■機能

strcat関数は文字列str1に文字列str2を連結します。
strncat関数では文字列str1に文字列str2をcount文字だけ連結します。結果文字列には'¥0'を付加します。

■例

文字列を連結し、結果を表示します。

```
#include <string.h>

main()
{
        static char str[40] = "Str 1";

        printf("before strcat:%s¥n", str);
        strcat(str, "Str 2");
        printf("after  strcat:%s¥n", str);
}
```

strcmp
stricmp
strcmpi

[・icmp]ver 4

■書式

```
#include <string.h>

int strcmp  (str1, str2);
int stricmp (str1, str2);
int strcmpi (str1, str2);

const char *str1;    文字列1
const char *str2;    文字列2
```

■戻り値

表の値を返します。

値	意味
< 0	str1 < str2
= 0	str1 = str2
> 0	str1 > str2

■機能

strcmp関数はstr1とstr2のふたつの文字列の先頭から文字列を比較し、辞書式順序での大小関係を返します。strcmpiまたはstricmp関数では、同じアルファベットの大文字と小文字は等しいと見なされます。

■ポイント

memicmp関数などと違い、文字列の終わりである'¥0'以降は比較しません。

stricmpとstrcmpiは、strcmpiがマクロで定義されていることを除けばまったく同じ機能です。

■例

標準入力から2行を受け取り比較します。日本語には対応していません。

```
#include <stdio.h>
#include <string.h>

void answer(int status);

main()
{
        char str1[30], str2[30];
        int status;

        printf("文字列1を入力して下さい¥n");
        gets(str1);
        printf("文字列2を入力して下さい¥n");
        gets(str2);

        status = strcmp(str1, str2);
        answer(status);
        status = stricmp(str1, str2);
        printf("大文字・小文字を無視した場合、");
        answer(status);
}

void answer(int status)
{
        printf("文字列1は文字列2");
        if (status < 0)
                printf("より小さい¥n");
        else if (status == 0)
                printf("と等しい¥n");
        else if (status > 0)
                printf("より大きい¥n");
}
```

strncmp
strncmpi
strnicmp

[・icmp]ver 4

■書式

```
#include <string.h>

int strncmp   (str1, str2, count);
int strncmpi  (str1, str2, count);
int strnicmp  (str1, str2, count);

const char *str1;    文字列1
const char *str2;    文字列2
size_t count;        文字数
```

■戻り値

次の値を返します。

値	意味
< 0	str1 < str2
= 0	str1 = str2
> 0	str1 > str2

■機能

strncmp関数はstr1とstr2のふたつの文字列の先頭からcount文字だけ比較し、辞書式順序で大小関係を返します。

文字列のうち、どちらかがcount文字より短い場合は、短い方の文字列が終わった時点で比較を中止します。

strncmpiまたはstrnicmp関数ではアルファベットの大文字と小文字を区別せず、等しいと見なします。

■ポイント

memicmp関数などと違い、文字列の終わりである'¥0'以降は比較しません。

strnicmpとstrncmpiは、strncmpiがマクロで定義されていることを除けばまったく同じ機能です。

■例

文字列先頭から指定された文字までを比較し、比較結果を表示します。

```
#include <stdio.h>
#include <string.h>

void answer(char *str, int cond);

main()
{
        char *str1, *str2;
        int count;

        str1 = "THIS IS TEST 1.";
        str2 = "This is test 2.";

        printf("何文字目まで比較しますか？¥n");
        scanf("%d", &count);

        printf("文字列１:%s¥n文字列２:%s¥n¥n", str1, str2);
        answer("strncmp", strncmp(str1, str2, count));
        answer("strnicmp", strnicmp(str1, str2, count));
}

void answer(char *str, int cond)
{
        printf("%s関数：", str);
        if (cond < 0)
                printf("文字列１の方が文字列２より小さい。¥n");
        else if (cond == 0)
                printf("文字列１と文字列２は等しい。¥n");
        else if (cond > 0)
                printf("文字列２の方が文字列１より小さい。¥n");
}
```

strcpy

■書式

#include <string.h>

char *strcpy (str1, str2);

char *str1; 文字列1
const char *str2; 文字列2

■戻り値

strcpy関数はstr1を返します。

■機能

strcpy関数は文字列str2の内容をstr1が指す領域に複写します。

■例

文字列を複写し、複写先の文字列の先頭1文字を'h'に変え、複写元・複写先ともに表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char str1[20], *str2;

        str2 = "Hello, world.";
        strcpy(str1, str2);
        str1[0] = 'h';
        puts(str1);
        puts(str2);
}
```

strncpy

■書式

#include <string.h>

char ＊strncpy (str1, str2, count)；

char ＊str1；　　　文字列1
const char ＊str2；文字列2
unsigned int count；文字数

■戻り値

str1を返します。

■機能

strncpy関数はstr2の内容をstr1に、先頭からcount文字を複写します。文字列str2がcount文字より短い場合は'¥0'を最後に付けます。長い場合は最後に'¥0'を付けません。

■ポイント

strncpyは、文字列str2がcount文字より長い場合にはstr1の最後に'¥0'を付加しません。したがって、このような場合の結果文字列にstrlen等の関数を適用すると正確な値を返さないことになります。

■例

文字列の先頭4文字を切り出して表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char buf[50], str1[50];

        printf("Who are you?¥n");
        gets(buf);
        strncpy(str1, buf, 4);
        str1[4] = '¥0';
        strcat(str1, ". How are you?");
        printf("Hello, %s¥n", str1);
}
```

strdup

■書式

#include <string.h>

char *strdup (str);

const char *str1;　　　文字列

■戻り値

領域が確保された場合、確保された領域、つまり文字列が複写された位置を指すポインタを返します。
領域が確保できなかった場合はNULLを返します。

■機能

strdup関数は「文字列strの長さ＋1」だけ領域を確保した後、内容を複写します。
メモリ不足等の理由で領域が確保できなかった場合はNULLを返します。

■ポイント

領域が「文字列の長さ＋1」だけ確保されるのは最後に'¥0'を付けるためです。複写するのはstrcpy関数と同じ'¥0'までです。

strdup関数は、内部的にmalloc関数を呼び出すので、複製された文字列が不要となった場合にはfree関数で領域を解放します。

■例

環境変数COMSPECの内容を複写した後、表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char *str;

        if ((str = getenv("COMSPEC")) {
                printf("COMSPECはセットされていません.¥n");
                exit(1);
        }
        if ((str = strdup(str)) == NULL) {
                printf("メモリが足りません.¥n");
                exit(2);
        }
        printf("現在のシェルは%sです.¥n", str);
        free(str);
        exit(0);
}
```

strlen

■書式

#include <string.h>

size_t strlen (str);

const char *str;　　　　文字列

■戻り値

文字列の長さを返します。

■機能

strlen関数は文字列strの長さを返します。文字列の長さとは、先頭から'¥0'の直前までの文字数です。'¥0'は文字数に含みません。

■例

入力された文字列の文字数を返します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char buf[128];

        printf("何か文字列を入力して下さい。¥n");
        gets(buf);
        printf("文字列は%d文字使用しています。", strlen(buf));
}
```

strset
strnset

■書式

#include <string.h>

char ＊strset (str, ch)；
char ＊strnset (str, ch, count)；

char ＊str；　　　文字列
int ch；　　　　　文字
unsigned int count；文字数

■戻り値

strを返します。

■機能

strset関数は文字列strのすべての文字を文字chに置き換えます。

strnset関数は文字列strの中の、先頭からcount文字だけを文字chに置き換えます。文字列strの長さがcount文字より短い場合は'¥0'の直前までが置き換えられます。

■ポイント

'¥0'は置き換わりません。

■例

入力された文字列を'＊'で囲み、表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char name[80], buf[80];

        gets(name);
        strcpy(buf, "*");
        strcat(buf, name);
        strcat(buf, "*");
        strcpy(name, buf);
        strset(name, '*');
        printf("%s¥n%s¥n%s¥n", name, buf, name);
}
```

strcspn

■書式

#include <string.h>

size_t strcspn (str1, str2)；

const char ＊str1； 文字列1
const char ＊str2； 文字列2

■戻り値

文字数を返します。

■機能

strspn関数は文字列1の先頭から文字列2の内容と比較し、文字列2の中のどれか1文字に該当する文字が(文字列1の中に)現れた時点で比較を終了し、先頭からその直前までの長さを返します。

■ポイント

strspnの逆の働きの関数です。左から右に文字列を比較するので、先頭から何文字目かを返す関数となります。

■例

標準入力から文字列を受け取り、先頭の単語の長さを表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        size_t length;
        char buf[128];
        printf("何か英文を入力して下さい。¥n");
        gets(buf);

        length = strcspn(buf, " ¥t¥n,.?!");
        printf("最初の単語の長さは%dです。", length);
}
```

strerror
_strerror

(strerror)ver 4,ver 5#

■**書式**

```
#include <string.h>

char *strerror (err);
char *_strerror (str);

int err;      エラー番号
char *str;    文字列
```

■**戻り値**

strerror関数はエラーメッセージの文字列を指すポインタを返します。

_strerror関数はユーザーの作成したエラーメッセージ+':'+' '+システムのエラーメッセージが格納された位置のポインタを返します。

■**機能**

sterror関数はエラー番号errに対するシステムのエラーメッセージ文字列を指すポインタを返します。

_strerror関数は、直前のエラーが格納されているerrnoの内容に対するエラーメッセージ文字列に、文字列strと":"を付加した文字列を指すポインタを返します。

■**ポイント**

strerrorはANSIのバージョンであり、_strerrorはMicrosoftの拡張バージョンです。

strpbrk

■書式

#include <string.h>

char ∗**strpbrk** (str1, str2)；

const char ∗str1； 文字列1
const char ∗str2； 文字列2

■戻り値

文字列1の中で文字列2中のいずれかの文字と最初に一致した文字へのポインタを返します。どれにも一致しなかった場合はNULLを返します。

■機能

strpbrk関数は文字列1の中に、文字列2の中のどれか1文字が存在するかどうか探索します。もしも存在する場合は、strpbrk関数は最初に文字が発見された位置のポインタを返します。もしも文字列1の中に文字列2の中の文字が1文字も発見されない場合はNULLを返します。

たとえば、

strpbrk ("Apr.8th 1988", ".,? ")；

では、"Apr.8th‥"の'.'の位置を指すポインタを返します。

ただし、ヌル文字は検索の対象となりません。

■ポイント

strcspn関数では発見された位置までの先頭からの文字数を返すのに対し、strpbrk関数では発見された位置を指すポインタを返します。

■例

標準入力から文字列を受け取り、先頭の単語を表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char *ptr, buf[128];

        printf("何か英文を入れて下さい。¥n");
        gets(buf);
        if ((ptr = strpbrk(buf, " ¥t¥r,.!?")) != NULL) {
                *ptr = '¥0';
                printf("最初の単語は、'%s'です。¥n", buf);
        }
}
```

strspn

■書式

#include <string.h>

size_t strspn (str1, str2);

const char *str1; 文字列1
const char *str2; 文字列2

■戻り値

文字列の長さを返します。

■機能

strspn関数は文字列1の先頭から文字列2の内容と比較し、文字列2の中のいずれの1文字にも該当しない文字が(文字列1の中に)現れた時点で比較を終了し、先頭からその直前までの長さを返します。

言い換えれば「文字列2の中のどれにもあてはまらない文字が一番最初に何文字目の次にあるか」を探す関数、ということになります。

■ポイント

strcspnの逆の働きの関数です。左から右に文字列を比較するので、先頭から何文字目かを返す関数となります。

■例

改行、スペースを読み飛ばし、それ以外の文字が現れる位置を表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        static char buf[] = "¥n¥n            This is sample.";

        printf("%s¥n", buf);
        printf("%d文字目から本文が始まります。¥n", strspn(buf, " ¥n") + 1);
}
```

strstr　ver 4

■書式

#include <string.h>

char ＊strstr (str1, str2)；

const char ＊str1； 文字列1
const char ＊str2； 文字列2

■戻り値

文字列2が文字列1の中に発見された場合、その部分の先頭を指すポインタが返されます。
発見されない場合はNULLが返されます。

■機能

strstr関数は、文字列2そのものが文字列1に存在するかを探索します。たとえば、文字列1が"Hello, C world."で文字列2が"world"とした場合、"world"は文字列1の中にあるので「存在する」ことになります。

文字列が存在する場合は、文字列1の中のその部分の最初の文字を指すポインタを返します。発見されなかった場合はNULLを返します。

■ポイント

文字列2が文字列1より長い場合はNULLを返します。

■例

入力に対し適当な文章を表示します。

```
#include <stdio.h>
#include <string.h>

struct voice {
        char *str1;
        char *str2;
};

struct voice table[] = {
        { "hello", "Oh, hello.  Nice to meet you." },
        { "how are you", "Thank you, I¥'m fine.  You too?" },
        { "how do you do", "How do you do.  I¥'m grad to meet you." },
        { "bye", "Good-bye.  See you again." },
        { "good morning", "Good morning, sir." },
        { "good evening", "Good evening.  It was so hot today, wasn¥'t it?" },
        { "good night", "Good night.  Have a nice dreaming." },
        { "your name?", "I don¥'t know what my name is." },
        { NULL, NULL }
};

main()
{
        char buf[100];
```

```
	struct voice *ptr;

	printf("何か英文をどうぞ。(例： hello!)¥n");
	gets(buf);
	strlwr(buf);
	for (ptr = table; ptr->str1 != NULL; ptr++) {
		if (strstr(buf, ptr->str1) != NULL) {
			printf("%s¥n", ptr->str2);
			break;
		}
	}
	if (ptr->str1 == NULL)
		printf("Sorry, I can¥'t understand what you say.¥n");
}
```

strtok

■書式

#include <string.h>

char ＊**strtok**（str1，str2）；

char ＊str1；　　文字列1
char ＊str2；　　文字列2

■戻り値

トークンが切り出せた場合には、切り出したトークンへのポインタを返します。トークンの切り出しに失敗した場合はNULLを返します。

■機能

str2をデリミタとして、str1からトークンを切り出します。次のトークンを切り出すためには、str1をNULLにして呼び出します。

■ポイント

この関数では日本語文字列を扱うことはできません。日本語文字列のトークン切り出しはjstrtok関数で行ってください。

■例

標準入力から入力された文字列の'.'',’や空白を区切りとして1行ずつ出力します。

```
#include <stdio.h>
#include <string.h>

static char buffer[1024];
char *dlm = "¥t¥n .,";

void main(void)
{
        char *p;

        while (fgets(buffer, sizeof(buffer), stdin) != NULL) {
                for (p = strtok(buffer, dlm);
                     p != NULL; p = strtok(NULL, dlm))
                        puts(p);
        }
}
```

strchr

■書式

#include <string.h>

char ＊strchr (str, key)；

const char ＊str； 文字列
int key； 文字

■戻り値

文字keyが文字列str中に発見された位置のポインタを返します。発見されなかった場合はNULLを返します。

■機能

strchr関数は文字列strの中から文字keyを探し、発見された場合はその位置を指すポインタを返します。文字列中に1文字もkeyが含まれていなかった場合はNULLを返します。

■ポイント

文字keyが文字列strの中に複数存在する場合は、最初に発見された文字の位置を指すポインタを返します。

■例

1文字を受け取り、入力された文字がアルファベット小文字の何番目にあるかを表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char *str = "abcdefghijklmnopqrstuvwxyz";
        char *ptr, str[40];
        int key;

        printf("何か文字を入力して下さい¥n");
        key = getch();
        if ((ptr = strchr(str, key)) == NULL)
                printf("その文字は含まれていません¥n");
        else
                printf("その文字は%d番目にあります¥n", (int)(ptr - str) + 1);
}
```

strrchr

■書式

#include <string.h>

char ＊strrchr (str, key)；

const char ＊str； 文字列
int key； 文字

■戻り値

文字列strの中で、文字keyが最後に発見された位置へのポインタを返します。発見されなかった場合はNULLを返します。

■機能

実際には、文字列strを後ろから検索し、文字keyが最初に発見された位置のポインタを返します。

■ポイント

strchr関数では文字列の先頭から検索を始めますが、strrchr関数では文字列末尾から検索を行います。

■例

文字列、文字を入力し、文字が文字列の中に現れる最後の位置を表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char *ptr, str[40];
        int key;

        printf("文字列を入力して下さい¥n");
        gets(str);
        printf("何か文字を入力して下さい¥n");
        key = getch();
        if ((ptr = strchr(str, key)) == NULL)
                printf("その文字は含まれていません¥n");
        else
                printf("一番最後のその文字は%d番目にあります¥n",
                                                (int)(ptr - str) + 1);
}
```

strrev

■書式

#include <string.h>

char ・strrev (str)；

char ・str；　　　文字列

■戻り値

strを返します。

■機能

strrev関数は文字列strの内容を逆さにします。'¥0'の位置は変わりません。

■ポイント

引数として渡された文字列は上書きされます。

■例

回文であるかどうかをチェックします。

```
#include <stdio.h>
#include <string.h>

void chgstr(unsigned char *str1, unsigned char *str2);

main()
{
        unsigned char buf1[50], buf2[50];
        static unsigned char buf[] = "Madam I'm Adam.";

        chgstr(buf1, buf);
        strcpy(buf2, buf1);
        strrev(buf2);
        printf("%s¥n", buf);
        if (strcmpi(buf1, buf2) == 0)
                printf("これは回文です¥n");
        else
                printf("これは回文ではありません¥n");
}

void chgstr(unsigned char *str1, unsigned char *str2)
{
        unsigned char c;

        while ((c = *str2++) != '¥0')
                if (strchr("、。¥"',.:? ", c) == NULL)
                        *str1++ = c;
}
```

strlwr
strupr

■書式

#include <string.h>

char ＊**strlwr** (str)；
char ＊**strupr** (str)；

char ＊str；　　　文字列

■戻り値

変換された文字列strを返します。

■機能

strlwr関数は文字列strの中の英大文字を英小文字に変換します。
strupr関数は逆に、文字列strの中の英小文字を英大文字に変換します。
いずれの関数も他の文字には影響しません。

■ポイント

strlwrやstrupr関数は日本語文字を考慮していません。ですから、Shift-JISコードの2バイト目に英文字がくる場合は注意が必要です。

■例

文字列をすべて大文字、すべて小文字に変換して表示します。

```
#include <stdio.h>
#include <string.h>

main()
{
        char buf[128];

        printf("英文を入力して下さい。¥n");
        gets(buf);
        printf("小文字: %s¥n", strlwr(buf));
        printf("大文字: %s¥n", strupr(buf));
}
```

lfind
lsearch

ver 4

■書式

#include <search.h>

char ＊**lfind**（key, base, num, width, (cmp)（)）；
char ＊**lsearch**（key, base, num, width, (cmp)（)）；

char ＊key；	検索するキー
char ＊base；	対象となる配列の先頭
unsigned ＊num；	配列の要素数
unsigned width；	一要素のデータ長
int（＊cmp)(elem1，elem2)；	比較関数
const void ＊elem1，＊elem2；	比較する要素

■戻り値

検索対象が見つかったら、見つかった要素へのポインタを返します。見つからない場合、lfind関数はNULLを返し、lsearch関数は配列の最後の要素(追加された要素)へのポインタを返します。

■機能

配列データから、ある要素を検索するのに使用します。

対象となる配列データはbaseを先頭とし、widthバイトの要素がnum個あると見なします。比較するときの判断基準としてcmpに指定された関数を呼び出し、戻り値が0だった場合に一致したと見なします。

lfind関数は見つからなければNULLを返すだけですが、lsearch関数は配列の最後にkeyの値を追加し、その値を返します。

■ポイント

検索対象はあくまでもwidthで指定した長さを持つ配列データで、どのような型のデータでも対象とすることができます。

lsearch関数で検索が一致しなかった場合は、最後の要素の次の場所に直接データが書き込まれるので注意してください。

■例

引数で指定された環境変数を見つけて表示するプログラムです。

```
#include <stdio.h>
#include <search.h>
#include <string.h>
#include <stdlib.h>

static int len;

static int cmp(char **s1, char **s2)
{
        int i;

        if ((i = strchr(*s2, '=') - *s2) != len)
                return 1;
        return strncmp(*s1, *s2, i);
}

void main(int ac, char **av, char **envp)
{
        int i;
        char *p, **pp;

        for (i = 0, pp = envp; *pp != NULL; pp++)
                i++;
        while (--ac) {
                len = strlen(strupr(p = *++av));
                if ((pp = (char **)lfind((char *)&p,
                                (char *)envp, &i,
                                sizeof (char *), cmp)) != NULL)
                        printf("%s¥n", *pp);
        }
        exit(0);
}
```

ソート・サーチ関数

bsearch

■書式

#include <search.h>　　ANSI準拠ならstdlib.h

char ＊bsearch (key, base, num, width, (cmp) ())；

const void ＊key；	検索するキー
const void ＊base；	検索対象となる配列の先頭
size_t num；	配列の要素数
size_t width；	各要素のサイズ
int (＊cmp) (elem1, elem2)；	比較関数
const void ＊elem1, ＊elem2；	比較する要素

■戻り値

検索対象が最初に見つかった場所へのポインタを返します。見つからなければNULLを返します。

■機能

配列データから、バイナリサーチの手法により、ある要素を検索するのに使用します。

対象となる配列データはbaseを先頭とし、widthバイトの要素がnum個あると見なします。

各要素の比較は、cmpとして指定された関数にふたつの要素へのポインタを渡して呼び出し、その戻り値によって次のように判断します。

値	判断
＜0	elem1 ＜ elem2
＝0	elem1 ＝ elem2
＞0	elem1 ＞ elem2

■ポイント

バイナリサーチを行うので、対象となる配列データはソート済みでなければなりません。

■例

標準入力から1000行までの文字列を読み込み、ソートして、各行の先頭が引数で指定された文字列と一致する最初の行を表示するプログラムです。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define PBUF    1000

static char *pbuf[PBUF];
static int pnum;
static int keylen;
```

```
int cmp(char **p1, char **p2)
{
        return strcmp(*p1, *p2);
}

int bcmp(char *key, char **data)
{
        return strncmp(key, *data, keylen);
}

void main(int ac, char **av)
{
        char buf[256], **r;

        for (pnum = 0; pnum < PBUF && gets(buf); pnum++)
                if ((pbuf[pnum] = strdup(buf)) == NULL) {
                        fprintf(stderr, "Too many data¥n");
                        break;
                }
        qsort(pbuf, pnum, sizeof (char **), cmp);
        while (--ac) {
                keylen = strlen(*++av);
                if ((r = bsearch(*av, pbuf, pnum,
                                sizeof (char *), bcmp)) != NULL)
                        puts(*r);
        }
        exit(0);
}
```

qsort

■書式

#include <search.h>　　ANSI準拠ならstdlib.h

void qsort (base, num, width, (cmp) ())；

void ＊base；	対象となる配列の先頭
size_t num；	配列の要素数
size_t width；	各要素のサイズ
int (＊cmp) (elem1, elem2)；	比較関数
cons void ＊elem1, ＊elem2；	比較する要素

■戻り値

ありません。

■機能

配列データの要素をクイックソートの手法によりソートします。対象となる配列データはbaseを先頭とし、各要素がwidthバイトの大きさを持つものと見なします。

各要素の比較は、cmpとして指定された関数にふたつの要素へのポインタを渡して呼び出し、その戻り値によって次のように判断します。

値	判断
＜ 0	elem1 ＜ elem2
＝ 0	elem1 ＝ elem2
＞ 0	elem1 ＞ elem2

■ポイント

配列の1要素の大きさが設定でき、比較関数の定義によって比較基準を定義できるので、実質的には、メモリ上に配置されたあらゆる型の配列データをソートすることが可能です。

■例

標準入力から1000行までの文字列を入力し、昇順に並べ替えて標準出力に出力するプログラムです。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define PBUF    1000

static char *pbuf[PBUF];
static int pnum;

int cmp(char **p1, char **p2)
```

```
{
        return strcmp(*p1, *p2);
}

void main(int ac, char **av)
{
        int i;
        char buf[256];

        for (pnum = 0; pnum < PBUF && gets(buf); pnum++)
                if ((pbuf[pnum] = strdup(buf)) == NULL) {
                        fprintf(stderr, "Too many data¥n");
                        break;
                }
        qsort(pbuf, pnum, sizeof (char **), cmp);
        for (i = 0; i < pnum; i++)
                puts(pbuf[i]);
        exit(0);
}
```

asctime

■書式

＃include ＜time.h＞

char ＊asctime (time)；

struct tm ＊time；　　　構造体へのポインタ

■戻り値

時刻を表わす文字列へのポインタを返します。

■機能

tm構造体に格納された時間を文字列に変換します。

■ポイント

通常gmtime関数やlocaltime関数の戻り値を使用します。

■例

システム時刻を表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;

        time(&t);
        printf("%s", asctime(localtime(&t)));
}
```

time

■書式

#include <time.h>

time_t time (t)

time_t *t;　　時刻を格納するバッファ

■戻り値

システム時刻を秒数で返します。

■機能

システム時刻を得ます。

■ポイント

timezone変数によって内部で調整されます。

■例

システム時刻を表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;

        time(&t);
        printf("%s", ctime(&t));
}
```

時間関数

ctime　ver 5#

■書式

＃include ＜time.h＞

char ＊ctime（t）；

time_t ＊t；　時刻を格納するバッファ

■戻り値

時刻を表わす文字列へのポインタを返します。

■機能

time_t型の変数に格納されている時刻を文字列に変換します。

■ポイント

通常time関数によって得られた値を使用します。

■例

システム時刻を表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;

        time(&t);
        printf("%s", ctime(&t));
}
```

gmtime

ver 5#

■書式
#include ＜time.h＞

struct tm ＊gmtime（t）；

time_t ＊t；　　時刻を格納するバッファ

■戻り値
time_t型から変換されたtm構造体へのポインタを返します。

■機能
time_t型の変数に格納されている時刻をtm構造体に変換します。

■ポイント
MS-DOSでは1980年以前の日付に対応できないため、それ以前の日付を指定するとNULLを返します。

■例
システム時刻をGMTで表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;

        time(&t);
        printf("%s", asctime(gmtime(&t)));
}
```

localtime　ver 5#

■書式

#include ＜time.h＞

struct tm ＊localtime (t)；

time_t ＊t；　　時刻を格納するバッファ

■戻り値

time_t型から変換されたtm構造体へのポインタを返します。

■機能

time_t型の変数に格納されている時刻をtm構造体に変換します。その時、必要に応じて地方時間やサマータイムを考慮します。

■ポイント

地方時間への変換はTZ環境変数の設定によります。TZ環境変数が設定されていない場合は、PST8PDSがデフォルトとなります。

■例

システム時刻を地方時間で表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;

        time(&t);
        printf("%s", asctime(localtime(&t)));
}
```

difftime　ver 4

■書式

＃include ＜time.h＞

double difftime (t1, t2)；

time_t t1；　時刻を格納するバッファ1
time_t t2；　時刻を格納するバッファ2

■戻り値

時間差を秒単位で返します。

■機能

ふたつの時間の差(t2 − t1)を計算します。

■例

ディスクへの100Kbytesの書き込みに要する時間を計測します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t1, t2;
        char buf[1024];
        FILE *fp;
        int i;

        time(&t1);
        printf("start: %s", ctime(&t1));
        if ((fp = fopen("test", "wb")) == NULL)
                abort();
        for (i = 0; i < 100; i++)
                if (fwrite(buf, sizeof buf, 1, fp) != 1)
                        abort();
        fclose(fp);
        time(&t2);
        printf("end:   %s", ctime(&t2));
        printf("diff: %f sec.¥n", difftime(t2, t1));
}
```

mktime　ver 5

■書式

#include ＜time.h＞

time_t mktime (tmp)

struct tm ＊tmp；　　構造体へのポインタ

■戻り値

time_t型の時間を返します。エラーの場合は(time_t)－1を返します。

■機能

tm構造体に格納されている時間をtime_t型に変換します。このときtm構造体のうち、tm_wdayとtm_ydayの値を正しく設定し直します。

■例

現在から10日後の時刻を表示します。mktime関数によって曜日も修正されるのがわかります。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;
        struct tm *tmp;

        time(&t);
        tmp = localtime(&t);
        tmp->tm_mday += 10;
        if (mktime(tmp) == (time_t)-1)
                printf("cannot make time¥n");
        else
                printf("%s", asctime(tmp));
}
```

tzset

■書式

```
#include <time.h>

void tzset (void);
```

■戻り値

ありません。

■機能

環境変数TZの設定に従ってdaylight, timezone, tznameの各グローバル変数を設定します。
TZが設定されていない場合は"PST8PDT"と見なします。

■ポイント

MS-DOSは内部的にローカル時間だけで管理されています。

■例

グリニッジ標準時との差を9時間であると見なし、地方時間と標準時間を表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        time_t t;

        putenv("TZ=JST-9");
        tzset();
        time(&t);
        printf("LOCAL: %s", asctime(localtime(&t)));
        printf("GMT:   %s", asctime(gmtime(&t)));
}
```

clock　　ver 5

■書式

#include <time.h>

clock_t clock (void);

■戻り値

プロセス時間(秒・CLK_TCK)を返します。計測できない場合は(clock_t)−1を返します。

■機能

この関数を実行するのに要した時間を計測します。最初に呼ばれた時からの経過時間を測定します。

■ポイント

(戻り値 / CLK_TCK)で秒に変換できます。

■例

5秒カウントします。

```
#include <stdio.h>
#include <time.h>

void main()
{
        clock_t t, t1;

        if ((t = clock()) == (clock_t)-1) {
                fprintf(stderr, "cannot get clock¥n");
                exit(1);
        }
        while (t < 5 * CLK_TCK)
                if ((t1 = clock()) != t)
                        printf("%d¥n", (int)((t = t1) / CLK_TCK));
        exit(0);
}
```

_strdate　ver 5

■書式

#include <time.h>

char •_strdate (buf)

char •buf;　　現在の日付

■戻り値

bufを返します。

■機能

現在の日付を「mm/dd/yy」の書式でbufに指定したバッファに書き込みます。

■ポイント

バッファは9バイト必要です。

■例

現在の日付を表示します。

```
#include <stdio.h>
#include <time.h>

void main()
{
        char buf[9];

        printf("%s¥n", _strdate(buf));
}
```

_strtime　　ver 5

■書式

#include <time.h>

char ・_strtime (buf)

char ・buf;　　時刻を表わす文字列

■戻り値

bufを返します。

■機能

現在の時刻を「hh:mm:ss」の書式でbufに指定されたバッファに書き込みます。

■ポイント

バッファは9バイト必要です。

■例

現在の時刻を表示します

```
#include <stdio.h>
#include <time.h>

void main()
{
        char buf[9];

        printf("%s¥n", _strtime(buf));
}
```

ftime

■書式

```
#include <sys/types.h>
#include <sys/timeb.h>
```

void ftime (timeptr);

struct timeb *timeptr;　格納域へのポインタ

■戻り値

ありません。

■機能

その時のシステム時刻を、timeptrで指定されたtimeb構造体に設定します。timeb構造体はsys/timeb.hに定義されており、次に示すフィールドを持ちます。

メンバ	内容
dstflag	サマータイムが設定されている場合には0以外の値を設定します。
time	グリニッジ標準時(GMT)を秒単位で設定します。
millitm	1秒未満の時間をミリ秒単位で設定します。
timezone	ローカルタイム(GMTとの差)を分単位で設定します。timezone(tzset参照)から算出されます。

■例

実行時の時刻を表示します。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/timeb.h>

void main()
{
        struct timeb tbuf;

        ftime(&tbuf);
        printf("%s", ctime(&tbuf.time));
}
```

utime

■書式

```
#include <sys/types.h>
#include <sys/utime.h>

int utime (path, times);

char *path;                    ファイル名
struct utimbuf *times;         時間の指定
```

■戻り値

成功なら0、エラーなら−1を返し、errnoに次の値を設定します。

記号定数	エラー内容
EACCES	パス名がディレクトリか、書き込み禁止のファイルです。
EINVAL	引数timesが不正です。
EMFILE	オープン中のファイルが多すぎます。
ENOENT	ファイルが見つかりません。

■機能

pathで指定したファイルのタイムスタンプをtimesが指す構造体の内容に従って変更します。utimbufはsys/utime.hで定義され、次に示すフィールドを持ちます。

メンバ	内容
actime	最終アクセス時間
modtime	最終更新時間

timesの代わりにヌルポインタを渡すと、現在の時間を設定します。

■ポイント

更新時間のみが意味を持ちます。アクセス時間は他の処理系との互換のために定義されています。

■例

指定ファイルのタイムスタンプを、すべて現在の時刻にします。

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/utime.h>

void main(ac, av)
int ac;
char **av;
{
        while (--ac) {
                if (utime(*++av, NULL) != 0) {
                        printf("cannot access '%s'¥n", *av);
                        exit(1);
                }
        }
        exit(0);
}
```

_clear87 ver 4

■書式

```
#include <float.h>

unsigned int _clear87 (void);
```

■戻り値

エラーリセット前の8087ステータスワードを返します。

■機能

8087(80287,80387)数値演算コプロセッサのエラーをクリアします。

以下の定数はfloat.hで定義され、ステータスワードとのビットANDを取ることによりエラーの発生状況を知ることができます。

記号定数	内容
SW_INVALID	無効演算
SW_DENORMAL	非正規化
SW_ZERODIVIDE	除算
SW_OVERFLOW	オーバーフロー
SW_UNDERFLOW	アンダーフロー
SW_INEXACT	精度の欠落

_control87　ver 4

■書式

```
#include <float.h>

unsigned int _control87 (new, mask)

unsigned int new;        コントロールワード
unsigned int mask;       マスク
```

■戻り値

8087コントロールワードを返します。

■機能

8087(80287, 80387)数値演算コプロセッサのコントロールワードを取得、セットします。

以下の定数はfloat.hで定義され、MCW_で始まる定数をmaskに、IC_, RC_, PC_で始まる定数をnewに設定し、_control87関数を呼び出せば8087の動作状態を変更することができます。

MCW_IC　無限大のコントロール

IC_AFFINE	連続モード
IC_PROJECTIVE	射影モード

MCW_RC　丸め(誤差処理)のコントロール

RC_CHOP	0方向への切捨て
RC_UP	切り上げ
RC_DOWN	切り捨て
RC_NEAR	近似値

MCW_PC　精度のコントロール

PC_24	仮数部24ビット長
PC_53	仮数部53ビット長
PC_64	仮数部64ビット長

この他、コントロールワードの初期状態の定数としてCW_DEFAULTが用意されています。

_fpreset　ver 4

■書式

```
#include <float.h>

void _fpreset (void);
```

■戻り値

ありません。

■機能

浮動小数点演算パッケージを初期化します。

_status87　　ver 4

■書式

```
#include <float.h>

unsigned int _status87 (void);
```

■戻り値

8087ステータスワードを返します。

■機能

8087 (80287 ,80387) 数値演算コプロセッサのステータスワードを取得します。
以下の定数はfloat.hで定義され、ステータスワードとのビットANDをとることによりエラーの発生状況を知ることができます。

記号定数	内容
SW_INVALID	無効演算
SW_DENORMAL	非正規化
SW_ZERODIVIDE	0除算
SW_OVERFLOW	オーバーフロー
SW_UNDERFLOW	アンダーフロー
SW_INEXACT	精度の欠落

acos

■書式

```
#include <math.h>

double acos (x);

double x;        数値
```

■戻り値

0からπまでの範囲でxのアークコサインを返します。

xは−1から1の間でなければいけません。xが範囲外の場合には、値として0が返されます。この場合errnoにはEDOM(演算の引数が不正)が設定され、stderrにDOMAINエラーメッセージが表示されます。

■機能

xのアークコサインを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

■ポイント

引数、戻り値ともにdouble型です。

asin

■書式

＃include <math.h>

double asin（x）;

double x;　　数値

■戻り値

$-\pi/2$から$\pi/2$の範囲でxのアークサインを返します。

xは－1から1の間でなければいけません。xが範囲外の場合には、値として0が返されます。この場合errnoにはEDOM（演算の引数が不正）が設定され、stderrにDOMAINエラーメッセージが表示されます。

■機能

xのアークサインを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

■ポイント

引数、戻り値ともにdouble型です。

atan

■書式

＃include ＜math.h＞

double atan（x）；

double x；　　数値

■戻り値

$-\pi/2$から$\pi/2$の範囲でxのアークタンジェントを返します。

■機能

xのアークタンジェントを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

■ポイント

引数、戻り値ともにdouble型です。

atan2

■書式

```
#include <math.h>

double atan2 (y, x);

double x;      数値
double y;      数値
```

■戻り値

−π/2からπ/2の範囲でy/xのアークタンジェントを返します。

x, yがともに0であった場合には値として0を返します。この場合errnoにEDOM(演算の引数が不正)が設定され、stderrにDOMAINエラーメッセージが表示されます。

■機能

y/xのアークタンジェントを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

■ポイント

引数、戻り値ともにdouble型です。

cabs

■書式

```
#include <math.h>

double cabs (z);

struct complex z;        数値
```

■戻り値

　複素数zの絶対値を返します。オーバーフローを起こした場合、errnoにERANGE(結果が範囲を越えている)が設定されます。

■機能

　複素数zの絶対値を求めます。複素数はmath.hによって定義されているcomplex型の構造体で、以下の通りです。

```
struct complex  {
        double x, y;
};
```

したがって次式とcabs関数は同値です。

```
sqrt(z.x * z.y + z.y * z.x)
```

■ポイント

　オーバーフロー時の戻り値は定義されていません。

ceil

■書式

```
#include <math.h>

double ceil (x);

double x;       数値
```

■戻り値

xを下回らない最小のdouble型の整数を返します。

■機能

xを下回らない最小の整数値を求めます。

■ポイント

エラー時の戻り値は定義されていません。

COS

■書式

```
#include <math.h>

double cos (x);

double x;        数値
```

■戻り値

xのコサインを返します。xの単位はラジアンです。xが非常に大きく、結果が無効な値の場合、stderrにTLOSSメッセージを表示して0を返します。

■機能

xのコサインを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

■ポイント

xが十分に大きい場合には結果の桁落ちが生じます。このときPLOSSエラーを生成しますが、エラーメッセージは表示されません。

また、xが非常に大きい場合にはTLOSSエラーをstderrに表示します。

いずれの場合もerrnoにERANGE(結果が長すぎる)が設定されます。

cosh

■書式

#include <math.h>

double cosh (x);

double x;　　　数値

■戻り値

xのハイパボリックコサインを返します。xの単位はラジアンです。
結果が大きすぎる場合はHUGE_VALを返し、errnoにERANGE(結果が長すぎる)が設定されます。

■機能

xのハイパボリックコサインを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

dieeetomsbin　　ver 4

■書式

#include ＜math.h＞

int dieeemstobin (src, dst)；

double ＊src；　変換する数値
double ＊dst；　変換された数値

■戻り値

成功した場合には戻り値として0を返します。オーバーフローが発生した場合には戻り値として1を返します。

■機能

IEEEフォーマットの倍精度浮動小数点数をマイクロソフトバイナリフォーマットへ変換します。
引数srcに変換するdouble型へのポインタを入れます。結果はdstによって示されたアドレスへ格納されます。

dmsbintoieee　　ver 4

■書式

#include ＜math.h＞

int dmsbintoieee (src, dst)；

double ＊src；　変換する数値
double ＊dst；　変換された数値

■戻り値

成功した場合には戻り値として0を返します。オーバーフローが発生した場合には戻り値として1を返します。

■機能

dieeetomsbin関数とは逆にマイクロソフトバイナリフォーマットの倍精度浮動小数点数をIEEEフォーマットに変換します。
引数srcに変換するdouble型へのポインタを入れます。結果はdstによって示されたアドレスへ格納されます。

exp

■書式

＃include <math.h>

double exp (x)；

double x；　　数値

■戻り値

xの指数関数を返します。オーバーフローが発生した場合には戻り値としてHUGE_VALを返し、errnoに以下の値が設定されます。またアンダーフローの場合には戻り値として0が返されますが、errnoは設定されません。

記号定数	エラー内容
ERANGE	結果が長すぎます。

■機能

xの指数関数を計算します。

fabs

■書式

#include <math.h>

double fabs (x)；

double x；　　数値

■戻り値

浮動小数点数xの絶対値を返します。エラー戻り値はありません。

■機能

浮動小数点数xの絶対値を返します。

fieeetomsbin　　ver 4

■書式

#include <math.h>

int fieeetomsbin (src, dst)；

float *src；　　変換する数値
float *dst；　　変換された数値

■戻り値

成功した場合には戻り値として0を返します。オーバーフローが発生した場合には戻り値として1を返します。

■機能

IEEEフォーマットの単精度浮動小数点数をマイクロソフトバイナリフォーマットへ変換します。
引数srcに変換するfloat型へのポインタを入れます。結果はdstによって示されたアドレスへ格納されます。

floor

■書式

＃include <math.h>

double floor（x）；

double x；　　数値

■戻り値

浮動小数点数の値を返します。エラー戻り値はありません。

■機能

xと等しいかxより小さいもののうち最大の整数を浮動小数点型で返します。

fmod

■書式

```
#include <math.h>

double fmod (x, y);

float x;      数値
float y;      数値
```

■戻り値

浮動小数点数の値を返します。yが0の場合には戻り値としてdouble型で0を返します。

■機能

x = iy + f（i:整数, x, f:同符号）の式を満たし、かつ、xの絶対値がyのそれより小さいという条件を満たすfをdouble型で返します。

■ポイント

yが0のときの戻り値0はdouble型です。

fmsbintoieee　ver 4

■書式
```
#include <math.h>

int fmsbintoieee (src, dst);

double *src;    変換する数値
double *dst;    変換された数値
```

■戻り値
成功した場合には戻り値として0を返します。オーバーフローが発生した場合には戻り値として1を返します。

■機能
fieeetomsbin関数とは逆にマイクロソフトバイナリフォーマットの単精度浮動小数点数をIEEEフォーマットに変換します。
引数srcに変換するdouble型へのポインタを入れます。結果はdstによって示されたアドレスへ格納されます。

frexp

■書式

♯include <**math.h**>

double frexp（x, exponential）;

double x；	数値
int ＊exponential；	指数部の数値

■戻り値

浮動小数点数xの仮数部を返します。xに0を指定すると0を返します。エラーを示す戻り値はありません。

■機能

浮動小数点数xについて、x＝m＊2＾nが成り立つような数m, nを求めます。仮数部mは戻り値として、指数部nはexponentialによって示されたアドレスに格納されます。引数xが0のとき、m, nは共に0となります。

■ポイント

mの絶対値は0.5以上1.0未満となります。

hypot

■書式

```
#include <math.h>

double hypot (x, y);

double x;      数値
double y;      数値
```

■戻り値

直角三角形の斜辺の長さを返します。オーバーフローが起きた場合はHUGE_VALを返し、errnoにERANGE(結果が長すぎる)が設定されます。

■機能

x, yの二辺がなす角が直角である三角形の斜辺の長さを計算します。したがって以下の式と同値です。

```
sqrt(x * x + y * y)
```

bessel

■書式

```
#include <math.h>

double j0 (x);
double j1 (x);
double jn (n, x);
double y0 (x);
double y1 (x);
double yn (n, x);

double x;        非負の浮動小数点数
int n;           関数の次数
```

■戻り値

浮動小数点数xに関するベッセル関数の値を返します。

y0,y1,ynに関しては、xは正数でなければいけません。

xが負の場合は、errnoに以下の値が設定され、stderrにDOMAINエラーメッセージを表示し、戻り値_HUGE_VALを返します。

エラー処理関数matherrによってエラー処理を行うことができます。

記号定数	エラー内容
EDOM	演算の引数です。

■機能

j0は次数0の第1種ベッセル関数を求めます。

j1は次数1の第1種ベッセル関数を求めます。

jnは次数nの第1種ベッセル関数を求めます。

y0は次数0の第2種ベッセル関数を求めます。

y1は次数1の第2種ベッセル関数を求めます。

y2は次数nの第2種ベッセル関数を求めます。

ldexp

■書式

```
#include <math.h>

double ldexp (x, exp);

double x;    浮動小数点数
int exp;     指数
```

■戻り値

x*2^expの値を返します。オーバーフローが起きた場合は、±HUGE_VALを返し、errnoにERANGE(結果が長すぎる)が設定されます。

■機能

x * pow(2, exp)の値を計算します。

log　　ver 5#

■書式
#include <math.h>

double log (x);

double x;　　浮動小数点数

■戻り値
浮動小数点数xの自然対数を返します。
xが負の場合には_HUGE_VALを返します。この場合、errnoにはEDOMが設定され、DOMAINエラーメッセージがstderrに表示されます。
また、xが0の場合には_HUGE_VALを返し、errnoにはERANGEが設定され、SINGエラーメッセージがstderrに表示されます。

記号定数	エラー内容
EDOM	演算の引数です。
ERANGE	結果が長すぎます。

■機能
浮動小数点数xの自然対数を計算します。

log10 ver 5#

■書式

```
#include <math.h>

double log10 (x);

double x;        浮動小数点数
```

■戻り値

浮動小数点数xの常用対数を返します。

xが負の場合には_HUGE_VALを返します。この場合、errnoにはEDOMが設定され、DOMAINエラーメッセージがstderrに表示されます。

また、xが0の場合には_HUGE_VALを返し、errnoにはERANGEが設定され、SINGエラーメッセージがstderrに表示されます。

記号定数	エラー内容
EDOM	演算の引数です。
ERANGE	結果が長すぎます。

■機能

浮動小数点数xの常用対数を計算します。

matherr

■書式

```
#include <math.h>

int matherr (x);

struct exception x;        数値演算例外情報
```

■戻り値

エラー発生時は0を、エラー処理に成功した場合は0以外を返します。

■機能

数値演算ライブラリ関数によって発生するエラーを処理します。matherrは数値演算関数がエラーを発生すると必ず呼び出されます。これを使ってユーザーが様々なエラー処理をするために、独自に関数を定義することが可能です。

matherrが呼び出される場合、数値演算関数はmath.hで定義されたexception型の構造体をmatherrへ渡します。exception型の構造体は以下の通りです。

```
struct exception {
    int type;                 エラーのタイプ
    char *name;               エラーが起きた関数
    double arg1, arg2;        エラーを起こした引数
    double retval;            戻り値
};
```

typeは数値演算エラーのタイプを指定します。値は以下のいずれかです。

記号定数	エラー内容
DOMAIN	引数定義域エラー
SING	引数誤り
PLOSS	部分桁落ち
TLOSS	全桁落ち
OVERFLOW	オーバーフロー
UNDERFLOW	アンダーフロー

nameはエラーが発生した数値演算関数名を表わすASCIZ文字列へのポインタです。arg1, arg2にはエラーを起こした引数が設定されます。引数がひとつの場合は、arg1だけ設定されます。retvalはユーザーが自由に変更することができるエラーのディフォルトの戻り値です。

matherrが0を返した場合、errnoには対応するエラー値が設定され、stderrに対応するメッセージが表示されます。matherrが0以外を返した場合にはerrnoは現在の値そのままで、エラーメッセージも表示されません。

■ポイント

struct exceptionの各メンバの指すものに留意しましょう。
エラー処理をした場合には戻り値に気を付けてください。

modf

■書式

#include <math.h>

double modf (x, intptr)；

double x；　　　浮動小数点数
double ＊intptr；

■戻り値

浮動小数点数xの符号付き小数部を返します。エラー戻り値はありません。

■機能

浮動小数点数xを小数部と整数部に分けます。xの符号付き小数部が戻り値です。整数部はintptrに浮動小数点値として格納されます。

pow

■書式

```
#include <math.h>

double pow (x, y);

double x;  被乗数
double y;  乗数
```

■戻り値

xのy乗を返します。被乗数x乗数yが共に0、もしくは被乗数が負で乗数が非整数の場合、errnoにEDOMが設定され、stderrにDOMAINエラーメッセージを出力して0を返します。被乗数が0で乗数が0の場合errnoにERANGEが設定され、戻り値としてHUGE_VALを返します。被乗数が非0で乗数が0の場合は戻り値として1を返します。オーバーフローが発生した場合、errnoにERANGEが設定され、戻り値として±HUGE_VALを返します。オーバーフローまたはアンダーフローが発生しても、エラーメッセージは表示されません。

記号定数	エラー内容
EDOM	演算の引数です。
ERANGE	結果が長すぎます。

■機能

xのy乗を計算します。

sin

■書式

```
#include <math.h>

double sin (x);

double x;       数値
```

■戻り値

　xのサインを返します。xの単位はラジアンです。xが非常に大きく、結果が無効な値の場合、stderrにTLOSSメッセージを表示して0を返します。

■機能

　xのサインを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

■ポイント

　xが十分に大きい場合には結果の桁落ちが生じます。このときPLOSSエラーが発生しますが、エラーメッセージは表示されません。

　また、xが非常に大きい場合にはTLOSSエラーをstderrに表示します。

　いずれの場合にもerrnoに以下の値が設定されます。

記号定数	エラー内容
ERANGE	結果が長すぎます。

sinh

■書式

#include <math.h>

double sinh (x);

double x;　　数値

■戻り値

xのハイパボリックサインを返します。xの単位はラジアンです。
結果が大きすぎる場合はHUGE_VALを返し、errnoにERANGE(結果が長すぎる)が設定されます。

■機能

xのハイパボリックサインを計算します。エラー処理関数matherrによってエラー処理を行うことができます。

sqrt

■書式

#include <math.h>

double sqrt (x);

double x;　　　非負の浮動小数点数

■戻り値

浮動小数点数xの平方根を返します。xが負数の場合、errnoに以下の値が設定され、stderrにDOMAINエラーメッセージを表示し、戻り値として0を返します。

エラー処理関数matherrによってエラー処理を行うことができます。

記号定数	エラー内容
EDOM	演算の引数です。

■機能

浮動小数点数xの平方根を計算します。

tan

■書式

```
#include <math.h>

double tan (x);

double x;        数値
```

■戻り値

xのタンジェントを返します。xの単位はラジアンです。xが非常に大きく、結果が無効な値の場合、stderrにTLOSSメッセージを表示して0を返します。

■機能

xのタンジェントを計算します。

■ポイント

xが十分に大きい場合には結果の桁落ちが生じます。このときPLOSSエラーが発生しますが、エラーメッセージは表示されません。

また、xが非常に大きい場合にはTLOSSエラーをstderrに表示します。

いずれの場合にもerrnoにERANGE(結果が長すぎる)が設定されます。

tanh

■書式

```
#include <math.h>

double  tanh (x);

double  x;        数値
```

■戻り値

xのハイパボリックタンジェントを返します。xの単位はラジアンです。エラー戻り値はありません。

■機能

xのハイパボリックタンジェントを計算します。

va_start
va_arg
va_end

ANSI

■書式

#include <stdarg.h>

void va_start (ap, v) ;
type va_arg (ap, type) ;
void va_end (ap) ;

va_list ap ;	引数リスト
v;	第1引数の前の引数
type;	引数の型

■戻り値

va_start, va_endマクロは戻り値がありません。va_argマクロは第2引数type型の可変引数を返します。

■機能

これらはANSI規格案準拠の可変引数を扱うマクロ群です。

va_startマクロは最初の引数を取得します。

va_argマクロは次の引数を取得します。

va_endマクロは可変引数リストをリセットします。

■ポイント

これらのマクロはANSI規格案に準拠しています。同じ機能を実現できるものとしてvarargs.hで提供されるUNIX System Vコンパチブルのマクロがあります。

■例

関数putstringsは不定長の引数を取り、NULLが現れるまで連続してコンソールに出力します。ANSI規格案準拠の可変引数リストを用いています。

```
#include <stddef.h>
#include <conio.h>
#include <stdarg.h>

void putstrings(char *str, ...)
{
        va_list ap;
        char *p;

        for (va_start(ap, str), p = str; p != NULL; p = va_arg(ap, char *)) {
                cputs(p);
        }
}

void main(int argc, char **argv)
{
        putstrings("argv[0] is ", argv[0],
                ".  It is filename of this Program.¥r¥n", NULL);
}
```

可変長の引数の並び

va_start
va_arg
va_end

UNIX

■書式

#include <varargs.h>

void va_start (ap);
type va_arg (ap, type);
void va_end (ap);

va_list op;	引数リスト
va_dcl	va_alistの宣言
va_alist	呼び出された関数の引数の名前

■戻り値

va_start, va_endマクロは戻り値がありません。va_argマクロは第2引数type型の可変引数を返します。

■機能

これらはUNIXコンパチブルの可変引数を扱うマクロ群です。
va_startマクロは最初の引数を取得します。
va_argマクロは次の引数を取得します。
va_endマクロは可変引数リストをリセットします。
va_dclマクロは引数が可変であることを宣言するマクロで、関数名と最初の大括弧{の間に書かねばなりません。
va_alistはva_dclが用いる可変引数の引数名として用いられますので、関数の仮引数宣言部分に書かねばなりません。

■ポイント

これらのマクロはUNIX System Vに準拠しています。同じ機能を実現できるものとしてstdarg.hで提供されるANSI規格案準拠マクロがあります。

■例

関数putstringsは不定長の引数を取り、NULLが現れるまで連続してコンソールに出力します。UNIX System Vコンパチブルの可変引数リストを用いています。

```
#include <stddef.h>
#include <conio.h>
#include <varargs.h>

void putstrings(va_alist)
va_dcl
{
        va_list ap;
        char *p;

        va_start(ap);
        while ((p = va_arg(ap, char *)) != NULL)
                cputs(p);
}

void main(int argc, char **argv)
{
        putstrings("argv[0] is ", argv[0],
                ".  It is filename of this Program.¥r¥n", NULL);
}
```

assert　　ver 5#

■書式
#include <assert.h>

void assert (exp)；

exp　　式

■戻り値
ありません。

■機能
assertマクロは、式が偽(式の値が0)になるときにエラーメッセージを出力して、abort関数を起動します。式が真(0以外)の場合は何もしません。
エラーメッセージは次のように出力します。

Assertion failed:　<exp> ,file <file> ,line <line>

<exp>には指定された値が、<file>と<line>にはassertのあるソースファイル名とその行番号が表示されます。

■ポイント
assertマクロはデバッグのために使用します。プログラムの先頭で

#define NDEBUG

と定義するか、コンパイル時に/Dを定義すればコンパイラはassertを無視しますので、デバッグが終了したらこの値を定義するか、assertの記述を取り除くようにします。

■例

assertマクロのテストです。第1引数に0か0以外の数値を指定してください。

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <assert.h>

void    main(int argc, char **argv)
{
        if (argc != 2) {
                puts("第１引数に０か０以外を指定してください。");
                return;
        }

        assert(atoi(argv[1]));

        puts("プログラムは正常に終了します。");
}
```

スタンダード関数その他

setjmp
longjmp

■書式

#include <setjmp.h>

int setjmp (env)；
void longjmp (env, value)；

jmp_buf env；スタック環境格納変数
int value；　setjmpへの戻り値

■戻り値

setjmp関数からの最初の戻り値は0になります。以降は、longjmp関数使用時にvalueとして指定した値を返します。

■機能

setjmp関数とlongjmp関数を組み合わせることにより、関数内に限定されないグローバルなジャンプを実行できます。

setjmp関数は、その時のスタック環境を指定の変数に格納し、以降のlongjmp関数の実行によってsetjmp関数実行時の実行環境を復元します。setjmp関数の戻り値によって、スタック環境を保存したのか、longjmp関数によって制御が渡されたのかを判断します。

valueに0を指定した場合のsetjmp関数からの戻り値は、強制的に1になります。

■ポイント

プログラム全体のエラー回復処理を呼び出すための手段として、時々使用されます。

■例

指定ファイルの内容を表示するプログラムですが、エラー発生時は常にmain関数に戻り、次の処理を続行する例です。

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf env;

static void readfile(char *p)
{
        FILE *fp;
        char buf[BUFSIZ];

        if ((fp = fopen(p, "r")) == NULL)
                longjmp(env, 1);
        while (fgets(buf, sizeof buf, fp))
                fputs(buf, stdout);
        if (!feof(fp))
                longjmp(env, 2);
        fclose(fp);
```

```
}

void main(int ac, char **av)
{
        char *p;

        while (--ac) {
                p = *++av;
                if (setjmp(env) != 0) {
                        perror(p);
                        fcloseall();
                        continue;
                }
                readfile(p);
        }
        exit(0);
}
```

abs
labs

■書式

#include <stdlib.h>

int abs (n)；
long labs (l)；

int n；　　数値
long l；　　数値

■戻り値

どちらの関数も引数の絶対値を返します。エラーを示す戻り値はありません。

■機能

abs関数はint型の数値nの絶対値を返します。
labs関数はlong型の数値lの絶対値を返します。

■ポイント

double型の絶対値を求める場合には、math.hで宣言されているfabs関数を使用します。

■例

－100から100までの3乗を求め、元の数値と3乗値、それぞれの絶対値を表示します。

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
        int i;
        long l;

        for (i = -100; i <= 100; i++) {
                l = (long)i * (long)i * (long)i;
                printf("%4d %8ld %4d %8ld¥n", i, l, abs(i), labs(l));
        }
}
```

min
max

ver 4

■書式

#include <stdlib.h>

type min (a,b)；
type max (a,b)；

type a, b；　　比較する数値

■戻り値

minマクロは小さい方の値を返します。
maxマクロは大きい方の値を返します。

■機能

minマクロはふたつの数値を比較して、小さい方の値を返します。
maxマクロはふたつの数値を比較して、大きい方の値を返します。

■ポイント

typeはどのような型であってもかまいません。また、min/maxマクロは、ふたつの数値が等しい場合にはbを返します。

■例

標準入力から複数行を受け取り、最大値と最小値を出力します。

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

void main(void)
{
        static char buffer[128];
        int max_i = INT_MIN;
        int min_i = INT_MAX;
        int n;

        while (fgets(buffer, sizeof(buffer), stdin) != NULL) {
                n = atoi(buffer);
                max_i = max(n, max_i);
                min_i = min(n, min_i);
        }
        printf("%d %d¥n", max_i, min_i);
}
```

div　ver 5

■書式

```
#include <stdlib.h>

div_t div (numer, denom);

typedef struct {
        int quot;    商
        int rem;     剰余
} div_t;

int numer;           被除数
int denom;           除数
```

■戻り値

除算の結果をdiv_t型で返します。

■機能

div関数はnumer÷denomを計算し、stdlib.hで定義されているdiv_t型で返します。div_t型はint型のメンバquotおよびremで構成されています。quotには商が、remには剰余が返されます。0除算を行った場合は、エラーメッセージを表示してプログラムの実行を中止します。

■ポイント

ldiv関数と同じ動作ですが、ldiv関数はlong型、div関数はint型の除算を行います。

■例

ふたつのint数値の商と余りを表示します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
	int i1;
	int i2;
	div_t dt;

	if (argc != 3) {
		fputs("二つのintの商と余りを表示します¥n", stderr);
		return 1;
	}
	i1 = atoi(argv[1]);
	i2 = atoi(argv[2]);
	if (i2 == 0) {
		fputs("0で除算はできません¥n", stderr);
		return 2;
	}
	dt = div(i1, i2);
	printf("%d ÷ %d = %d ... %d¥n",
		i1, i2, dt.quot, dt.rem);
	return 0;
}
```

getenv

■書式

#include ＜stdlib.h＞

char ・getenv (var)；

const char ・var；　　　　環境変数名

■戻り値

指定された環境変数の内容を返します。指定された環境変数が未定義の場合は、NULLを返します。

■機能

getenv関数は環境変数の中からvarで指定された変数の内容を指すポインタを返します。この値は環境変数テーブルのエントリを指していますので、このポインタを使用して内容を変更してはいけません。

■ポイント

環境変数の内容を変更する場合はputenv関数を使用します。

■例

引数で指定された環境変数の値を表示します。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        char envname[40], *p;

        if (argc != 2) {
                fputs("環境変数の値を表示します。¥n", stderr);
                return 2;
        }
        strupr(strncpy(envname, argv[1], 39));
        p = getenv(envname);
        if (p == NULL) {
                printf("環境変数 ¥"%s¥" は設定されていません。¥n", envname);
                return 1;
        } else {
                printf("環境変数 ¥"%s¥" の値は ¥"%s¥" です。¥n", envname, p);
                return 0;
        }
}
```

ldiv　　ver 5

■書式

#include <stdlib.h>

ldiv_t ldiv (numer, denom)；

typedef **struct**　{
　　long quot；商
　　long rem；剰余
}　**ldiv_t**

long int numer；　被除数
long int denom；　除数

■戻り値

除算の結果をldiv_t型で返します。

■機能

ldiv関数はnumer÷denomを計算し、stdlib.hで定義されているldiv_t型で返します。ldiv_t型はlong型のメンバquotおよびremで構成されています。quotには商が、remには剰余が返されます。0除算を行った場合は、エラーメッセーラージを表示してプログラムの実行を中止します。

■ポイント

div関数と同じ動作ですが、div関数はint型、ldiv関数はlong型の除算を行います。

■例

ふたつのlong数値の商と余りを表示します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
	long l1;
	long l2;
	ldiv_t ldt;

	if (argc != 3) {
		fputs("二つのlong intの商と余りを表示します¥n", stderr);
		return 1;
	}
	l1 = atol(argv[1]);
	l2 = atol(argv[2]);
	if (l2 == 0L) {
		fputs("0で除算はできません¥n", stderr);
		return 2;
	}
	ldt = ldiv(l1, l2);
	printf("%ld ÷ %ld = %ld ... %ld¥n",
		l1, l2, ldt.quot, ldt.rem);
	return 0;
}
```

putenv

■書式

#include <stdlib.h>

int putenv (envstr);

char *envstr;　環境変数の設定文字列

■戻り値

正常にセットされた場合は0を返します。エラーの場合は−1を返します。

■機能

putenv関数は現プロセスの環境変数を設定します。文字列envstrは以下の書式にします。

```
environ=string
```

environは環境変数の名前であり、stringはその内容です。command.comのsetの書式と同一です。環境変数の変更は、必ずこのputenvを使用してください。

■ポイント

変更／追加された環境変数は現在のプロセス以下のプロセスでのみ有効です。putenv関数を用いて親プロセスの環境変数を変更／追加することはできません。そのため、たとえば

```
void main (void)
 {
        putenv ("PATH=a:¥usr¥bin;");
}
```

というプログラムを実行しても、実行後の環境は変更されません。

■例

環境変数を設定して、子プロセスのCOMMAND.COMを呼び出します。

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

int main(void)
{
        putenv("NEWENV=New Environment Variable");
        fputs("NEWENVという名前の新しい環境変数を作成して、", stdout);
        fputs("COMMAND.COMを起動します。SETコマンドで環境変数を確認した後、",
                stdout);
        fputs("EXITコマンドで終了してください¥n", stdout);

        spawnl(P_WAIT, getenv("COMSPEC"), "command", NULL);

        fputs("プロセスを終了します。プロセスが終了すると、", stdout);
        fputs("環境変数は元の状態に戻ります。", stdout);
        fputs("SETコマンドで環境変数を確認してください。", stdout);
        fputs("NEWENVは設定されていません。¥n", stdout);
        return 0;
}
```

rand

■書式

```
#include <stdlib.h>

int rand();
```

■戻り値

乱数を返します。エラー戻り値はありません。

■機能

乱数を発生させ、得られた乱数を返します。

■例

srand関数の例を参照してください。

srand

■書式

#include <stdlib.h>

void srand (seed)

unsigned int seed； 乱数の数値

■戻り値

ありません。

■機能

乱数発生のための数値をseedに初期化します。

■ポイント

乱数を発生させるrand関数はsrand関数で設定された数値を元に、順次、乱数を作成して行きます。そのため、srand関数を用いて任意の数値で初期化しない場合には、rand関数で得られる乱数列は同一のものになってしまいます。ユーザーからの入力を求めたり、time関数で得られた時刻情報、getpid関数で得られるプロセスIDを用いて初期化する等の手法が一般的です。

■例

まず、乱数を初期化せず10個の乱数を表示します。次に、getpid関数で取得されたプロセスIDによって乱数を初期化し、10個の乱数を表示します。最初の10個は毎回同じ結果となり、次の10個は実行ごとに違う数値をとることがわかります。

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

void putrand(void)
{
        int i;

        for (i = 0; i < 10; i++) {
                printf("%7d", rand());
        }
        printf("¥n");
}

void main(void)
{
        putrand();
        srand((unsigned int)getpid());
        putrand();
}
```

swab

■書式

＃include ＜stdlib.h＞

void swab (src, dst, nbyte)；

char ＊src； 元データ
char ＊dst； 変換後のデータ
int nbyte； 変換するバイト長

■戻り値

変換した結果をdstから格納する以外は、戻り値はありません。

■機能

swab関数はsrcからの内容をワードデータの並びと見て、上下バイトを交換してdstにnbyte分転送します。

■ポイント

バイト並びが異なる機械にデータを変換する目的で使用します。MS-DOSの動作する8086系のCPUでは1ワードは下位バイト・上位バイトの順でメモリ上に配置されますが、68000系のCPUでは上位バイト・下位バイトの順となります。

■例

8086系のバイト並びのHIGH, LOWバイトを交換し、交換前後の状態を表示します。

```
#include <stdio.h>
#include <stdlib.h>

int tab1[5] = {100, 200, 300, 400, 500};
int tab2[5];

void dump(unsigned char *p, int n)
{
        while (n-- > 0)
                printf("%02x ", *p++);
        putchar('¥n');
}

void main(void)
{
        dump((unsigned char *)tab1, 10);
        swab((char *)tab1, (char *)tab2, sizeof(tab1) * 2);
        dump((unsigned char *)tab2, 10);
}
```

_lrotl　_rotl
_lrotr　_rotr

ver 5

■書式

#include <stdlib.h>

```
unsigned long _lrotl (num, count);
unsigned long _lrotr (num, count);
unsigned int _rotl (inum, count);
unsigned int _rotr (inum, count);

unsigned long num;     数値
unsigned int inum;     数値
int count;             ローテートする回数
```

■戻り値

ローテートした結果を返します。エラー戻り値はありません。

■機能

_lrotl関数はnumを左方向(上位ビット方向)へcount回ローテートします。
_lrotr関数はnumを右方向(下位ビット方向)へcount回ローテートします。
_rotl関数はinumを左方向(上位ビット方向)へcount回ローテートします。
_rotl関数はinumを右方向(下位ビット方向)へcount回ローテートします。

_lrotlと_lrotr関数はunsigned long型の、_rotlと_rotr関数はunsigned int型の数値をローテートします。符号付きの数値をローテートすることはできません。

■ポイント

ローテートとは、数値を2進数にして考え、それぞれのビットを右もしくは左にずらして「回転」させることを言います。

■例

引数を16回右ローテートして、順次表示します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        int i;
        unsigned int n;
        char *p;

        if (argc != 2) {
                fputs("引数を１６回右ローテートして表示します。¥n", stderr);
                return 2;
        }
        n = (unsigned int)(strtoul(*++argv, &p, 16) & 0xffff);
        for (i = 0; i < 16; i++) {
                printf("%2d: %04x¥n", i, n);
                n = _rotr(n, 1);
        }
        return 0;
}
```

_searchenv　　ver 5

■書式

#include <stdlib.h>

void _searchenv (file, env, path)；

char ＊name；　検索するファイル名
char ＊env；　参照する環境変数名
char ＊path；　検索されたファイルのフルパス名

■戻り値

pathにフルパス名もしくはヌル文字のみの文字列をセットします。戻り値はありません。

■機能

_searchenv関数はファイルの検索を、指定された環境変数名に設定されているディレクトリの範囲で行います。発見できなかった場合は""('¥0'のみからなる文字列)をpathにセットし、発見した場合はファイル名を含むフルパス名をpathにセットします。

たとえば、_search("command.com", "PATH", fullname)；は、カレントディレクトリと環境変数PATHに設定されたディレクトリの中からcommand.comを検索します。

環境変数にはPATH=a:¥bin；a:¥usr¥binなどのように、複数のディレクトリを指定することができます。

■ポイント

検索の順序は、カレントディレクトリ→環境変数のディレクトリです。path［0］ == '¥0'は発見されなかったことを意味します。

■例

環境変数PATHに従ってコマンドファイルを検索します。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
        char buffer[128];

        if (argc == 1) {
                fputs("コマンドの格納されているディレクトリを表示します¥n",
                        stderr);
                return 2;
        }
        while (--argc > 0) {
                _searchenv(*++argv, "PATH", buffer);
                if (*buffer == '¥0')
                        printf("%s は見つかりません。¥n", *argv);
                else
                        printf("%s が見つかりました。¥n", buffer);
        }
        return 0;
}
```

perror

■書式
＃include <stdio.h>

void perror（str）;

const char ＊str;　　文字列

■戻り値
ありません。

■機能
perror関数はerrnoにセットされている直前のエラーに対応したメッセージをコンソールに出力します。その際に、指定された文字列と':'を付加して出力します。

perror関数で表示されるエラーメッセージは、変数sys_errlistを通して得られます。

■ポイント
正確にエラーを表示するためには、エラーが起こった直後にperrorを実行します。

■例
$$$.$$$ というファイルをオープンし、存在しない場合にはperrorを用いてエラーメッセージを表示します。

```
#include <stdio.h>

char *fil = "$$$.$$$";  /* 存在しないファイル */

main()
{
        FILE *fp;

        if ((fp = fopen(fil, "r")) == NULL)
                perror(fil);
        else
                fclose(fp);
}
```

iskana
iskpun
iskmoji

■書式

#include <jctype.h>

int iskana (c);
int iskpun (c);
int iskmoji (c);

int c; 調べる文字

■戻り値

それぞれの条件と文字を照らし合わせ、条件に合う場合は0以外を、合わない場合は0を返します。

■機能

これらはjctype.hにてマクロで定義されています。
iskanaは文字cが半角カナであるかどうか調べます。
iskpunは文字cが半角カナの句読点であるかを調べます。
iskmojiは文字cが半角カナそのもの(濁点などを含まない)であるかどうか判断します。

■ポイント

条件と合致しない場合は0を返しますが、条件と合致する場合の戻り値については「0以外」としか定義されていません。

■例

文字を句読点で区切って表示、改行します。

```
#include <stdio.h>
#include <jctype.h>

main()
{
        int c;
        unsigned char *ptr;
        ptr = "ﾓｼﾞ ｦ､ ｸﾄｳﾃﾝ ﾃﾞ ｸｷﾞｯﾃ､ ﾋｮｳｼﾞ､ｶｲｷﾞｮｳ ｼﾏｽ｡";

        puts(ptr);
        while ((c = *ptr++) != '¥0') {
                putchar(c);
                if (iskpun(c) != 0)
                        putchar('¥n');
        }
}
```

isalkana　isprkana
ispnkana　isgrkana
isalnmkana

■書式

```
#include <jctype.h>

int isalkana (c);
int ispnkana (c);
int isalnmkana (c);
int isprkana (c);
int isgrkana (c);

int c;             調べる文字
```

■戻り値

それぞれの条件と文字を照らし合わせ、条件に合う場合は0以外を、合わない場合は0を返します。

■機能

これらはjctype.hにてマクロで定義されています。
isalkana(c)はisalpha(c) || iskmoji(c)と同じで、cが英文字かカナ文字かを判断します。
ispnkana(c)はispunct(c) || iskpun(c)と同じで、英句読点とカナ句読点を判断します。
isalnmkana(c)はisalnum(c) || iskanji(c)と同じで、英文字か漢字文字かを判断します。
isprkana(c)はisprint(c) || iskana(c)と同じで、表示可能文字であるかを判断します。
isgrkana(c)はisgrph(c) || iskana(c)と同じで、空白文字を含め、表示可能文字かどうかを判断します。

■ポイント

条件と合致しない場合は0を返しますが、条件と合致する場合の戻り値については「0以外」としか定義されていません。

■例

英文字列、1バイトカナ文字列を句読点で区切って改行します。

```
#include <stdio.h>
#include <jctype.h>

main()
{
        int c;
        unsigned char *ptr;
        ptr = "English string, ﾔ､ｶﾅ ﾌﾞﾝ ｦ､ｸﾄｳﾃﾝ ﾃﾞ ｸｷﾞｯﾃ ｶｲｷﾞｮｳ ｼﾏｽ｡":

        puts(ptr);
        while ((c = *ptr++) != '¥0') {
                putchar(c);
                if (ispnkana(c) != 0)
                        putchar('¥n');
        }
}
```

iskanji
iskanji2

■書式

#include <jctype.h>

int iskanji (c);
int iskanji2 (c);

int c;　　　　調べる文字

■戻り値

それぞれの条件と文字を照らし合わせ、条件に合う場合は0以外を、合わない場合は0を返します。

■機能

これらはjctype.hにてマクロで定義されています。
iskanjiは文字cが全角文字のシフトJISコードの第1バイトとして正しいかどうかを判断します。
iskanji2は文字cが全角文字のシフトJISコード第2バイトとして正しいかどうかを判断します。

■ポイント

条件と合致しない場合は0を返しますが、条件と合致する場合の戻り値については「0以外」としか定義されていません。

■例

全角文字の文字列中の文字数を表示します。

```
#include <stdio.h>
#include <jctype.h>

main()
{
        unsigned char *ptr;
        int len;

        ptr = "全角文字のstring中の文字数をcheckします。";
        len = 0;

        puts(ptr);
        while (*ptr != '¥0') {
                if (iskanji(*ptr) != 0) {
                        len++;
                        if (!*++ptr)
                                break;
                }
                ptr++;
        }
        printf("全角文字は%d文字でした。¥n", len);
}
```

btom

■書式

#include <jstring.h>

int btom (str, nbyte)；

const unsigned char ＊str；文字列
int nbyte；　　　　　バイト数

■戻り値

文字列strのnbyteバイト目までの文字数を返します。

■機能

btom関数は文字列strで指し示すバッファのnbyteバイト目までに、文字が何文字あるかを数えます。普通のキャラクタは1文字、全角文字は2バイトで1文字として数えます。

■ポイント

nbyteバイト目より前に'¥0'が現れた場合(つまり文字列strがnbyteより短かかった場合）はその直前までの文字数を返します。

nbyteバイト目がちょうど全角文字の1バイト目にあたる場合は、その全角文字は数えられません。全角文字の2バイト目が'¥0'の場合も無視されます。

■例

漢字混じり文字列の先頭から20バイト目までの文字数を表示します。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char buf[128];
        int moji;

        printf("漢字(全角文字)混じりの文を入力して下さい。¥n");
        gets(buf);
        moji = btom(buf, 20);

        if (moji == 0)
                printf("何も入力されませんでした。¥n");
        else
                printf("%d文字入力されました。¥n", moji);
}
```

chkctype

■書式

```
#include <jstring.h>

int chkctype (c, type);

unsigned char c;        調べる文字
int type;               機能
```

■戻り値

以下の値を返します。

typeの値	戻り値	意味
1以外	CT_ANK	ANK文字
1以外	CT_KJ1	漢字の第1バイト目
1	CT_KJ2	漢字の第2バイト目
1	CT_ILGL	無効な文字

CT_ANK, CT_KJ1などの記号定数はjctype.hで定義されています。

■機能

chkctype関数は文字cがどのような種類の文字であるかを調べます。指定された文字がANKもしくは全角文字の第1バイトであるか、または全角文字の第2バイトとして正しい値であるかどうかなどの、文字のタイプを調べることができます。たとえば、現在ポインタが指している位置の文字は全角文字であるのか、などの判断を行うことができます。

■ポイント

文字cが'¥0'の場合は無条件にCT_ILGLを返します。

■例

漢字の前に「/」マークを付けます。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr1, *ptr2;

        ptr1 = "This is jstrskip関数's test.";
        ptr2 = ptr1;
        while (*ptr2 != '¥0') {
                if (chkctype(*ptr2, 0) == CT_KJ1) {
                        putchar('/');
                        putchar(*ptr2++)';
                }
                putchar(*ptr2++);
        }
}
```

hantozen

■書式

＃include ＜jstring.h＞

unsigned short hantozen (c)；

unsigned short c；　　変換する文字

■戻り値

文字cを全角文字に変換し、変換された全角文字コードを返します。ただし、文字cが全角文字に変換できない場合は、cがそのまま返されます。

■機能

hantozen関数は半角文字cを全角文字に変換します。(例：8→８)

■ポイント

ASCIIコードの0x20から0x7eまでしか変換されません。

■例

半角文字列を全角に変換して表示します。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char buf[128], *ptr;
        unsigned short c;

        puts("半角で文を入力して下さい。");
        gets(buf);

        ptr = buf;

        while (*ptr != '¥0') {
                c = hantozen((unsigned short)(*ptr++));
                putchar(c >> 8);
                putchar(c & 0xff);
        }
}
```

jisalpha jisupper jislower jisdigit jiskata jishira jiskigou jisspace

■書式

```
#include <jstring.h>

int jisalpha (c);
int jisupper (c);
int jislower (c);
int jisdigit (c);
int jiskata (c);
int jishira (c);
int jiskigou (c);
int jisspace (c);

unsigned short c;        調べる文字
```

■戻り値

それぞれの条件と文字を照らし合わせ、条件に合う場合は0以外を、合わない場合は0を返します。

■機能

これらの関数は文字cがどのような種類の全角文字であるかを調べます。文字cが全角文字ではない場合には0を返します。

jisalphaは文字cが全角のアルファベットであるかどうか調べます。
jisupperは文字cが全角のアルファベットの大文字であるかどうか調べます。
jislowerは文字cが全角のアルファベットの小文字であるかどうか調べます。
jisdigitは文字cが全角の数字であるかどうか調べます。
jiskataは文字cが全角のカタカナであるかどうか調べます。
jishiraは文字cが全角のひらがなであるかどうか調べます。
jiskigouは文字cが全角の記号文字であるかどうか調べます。
jisspaceは文字cが全角のスペース(Shift-JISコードで0x8140)かどうか調べます。

■ポイント

is..と異なり、文字cには通常全角文字を表わすShift-JISコードを与えます。

■例

文字列の中のひらがなの文字数を数えます。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr;
        int c;
        c = 0;
        ptr = "文字列の中のひらがなの文字数を数えます。";

        puts(ptr);
        while (*ptr != '¥0') {
                if (jishira((*ptr << 8) | *(ptr + 1)) != 0)
                        c++;
                ptr += 2;
        }
        printf("ひらがなは%d文字ありました。¥n", c);
}
```

jmstojis
jistojms

■書式

#include <jstring.h>

unsigned short jmstojis (c);
unsigned short jistojms (c);

unsigned short c;　　　変換するコード

■戻り値

jmstojisはShift-JISコードとして与えられているcをJISコードに変換した値を返します。
jistojmsはJISコードとして与えられているcをShift-JISコードに変換した値を返します。
いずれの関数も変換できない値の場合は0を返します。

■機能

これらの関数はShift-JISコード(マイクロソフト漢字コード)とJISコードを相互変換します。
jmstojisはShift-JISコードをJISコードに変換します。
jistojmsはJISコードをShift-JISコードに変換します。

■ポイント

どちらの関数も変換できない場合は0を返します。

■例

全角文字「A」から「Z」までのJIS漢字コードを16進数で表示します。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned short c, d;

        for (c = 'A'; c <= 'Z'; c++) {
                d = hantozen(c);
                putchar(d >> 8);
                putchar(d & 0xff);
                printf(" %04X¥t", jmstojis(d));
        }
}
```

jiszen　jisl1　jisl0　jisl2　ver 4

■書式

```
#include <jstring.h>

int jiszen (c);
int jisl0 (c);
int jisl1 (c);
int jisl2 (c);

unsigned short c;　　　　調べる文字
```

■戻り値

それぞれの条件と文字を照らし合わせ、条件に合う場合は0以外を、合わない場合は0を返します。

■機能

jiszenは文字cが全角文字かどうかを調べます。
jisl0は文字cが全角のひらがな、カタカナ、英数字などであるかどうかを調べます。
jisl1は文字cが第1水準の全角文字であるかどうか調べます。
jisl2は文字cが第2水準の全角文字であるかどうか調べます。

■ポイント

is..の関数と違い、文字cには全角文字を指定できます。全角文字を指定する場合、以下のように'でくくって指定してはいけません。

```
jiszen('字');
```

この場合は正しいShift-JISコードになりませんので、以下のような形にします。

```
jiszen (0x8e9a);
```

■例

JIS第1水準文字、JIS第2水準文字の文字数を数えます。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr;
        unsigned short c;
        int l1, l2;

        ptr = "璽鼈・これは第２水準です。";
        l1 = l2 = 0;

        while (*ptr != '¥0') {
                c = (*ptr << 8) | *(ptr + 1);
                ptr += 2;
                if (jisl1(c) != 0)
                        l1++;
                if (jisl2(c) != 0)
                        l2++;
        }

        printf("第１水準漢字:%d文字、第２水準漢字:%d文字¥n", l1, l2);
}
```

jstradv

■書式

#include <jstring.h>

unsigned char ＊jstradv (str, nmoji)；

const unsigned char ＊str；対象となる文字列
int nmoji；　　　　　　　文字数

■戻り値

文字列strの先頭よりnmoji文字分進んだ位置を指すポインタを返します。

■機能

jstradv関数は〝文字数〟を単位として、文字列strの先頭よりnmoji文字分進めた位置を指すポインタを計算します。

通常のポインタの計算(str＋＋；など)では全角文字を考慮に入れていません。たとえば、現在「例」という全角文字の1バイト目をポインタptrが指しているとすると、＊(＋＋ptr)は「例」という文字の2バイト目を指してしまうことになります。

jstradv関数は文字列strの先頭よりnmoji文字分だけ進んだポインタを計算し、返します。

nmoji文字目より前に'¥0'が現れた場合、jstradv関数はその'¥0'を指すポインタを返します。

■ポイント

途中の全角文字の2バイト目が'¥0'だった場合、jstradv関数は1バイト目も'¥0'であると見なし、その文字の1バイト目を指すポインタを返します。

■例

文字の間にスペース(0x20)を挿入します。

```
#include <stdio.h>
#include <jctype.h>
#include <jstring.h>

main()
{
        unsigned char *ptr;

        ptr = "jstradv関数は文字単位でﾎﾟｲﾝﾀを進めます。";

        while (*ptr != '¥0') {
                putchar(*ptr);
                if (chkctype(*ptr, 0) == CT_KJ1)
                        putchar(*(ptr + 1));
                putchar(' ');
                ptr = jstradv(ptr, 1);
        }
}
```

jstrchr

■書式

#include <jstring.h>

unsigned char ・jstrchr (str, c)；

unsigned char ・str；
unsigned short c；

■戻り値

文字が発見されたら、その文字を指すポインタを返します。発見できなかった場合は、NULLを返します。

■機能

jstrchr関数はstrchr関数の日本語対応版です。文字列strの中で、文字cが含まれる位置へのポインタを返します。文字cはcharでなくunsigned shortとして与えます。全角文字を探す場合はcに、2バイトのデータをShift-JISコードで直接代入します。

■例

文字に句読点が含まれるかどうかをチェックします。

```
#include        <stdio.h>
#include        <jstring.h>

main()
{
        unsigned char *ptr;
        ptr = "これは、jstrchr関数のサンプルです。";

        printf("文字列:%s¥n", ptr);

        if (jstrchr(ptr, 0x8141) != NULL)
                printf("読点があります。¥n");
        if (jstrchr(ptr, 0x8142) != NULL)
                printf("句点があります。¥n");
}
```

jstrcmp

■書式

#include <jstring.h>

int jstrcmp (str1, str2);

unsigned char *str1; 文字列1
unsigned char *str2; 文字列2

■戻り値

表の値を返します。

値	意味
< 0	str1 < str2
= 0	str1 = str2
> 0	str1 > str2

■機能

jstrcmp関数はstrcmp関数の日本語対応版です。文字の大小関係は以下のようになります。

ANK文字<半角カナ文字<全角文字

■ポイント

文字列中の全角文字の2バイト目が'¥0'の場合は1バイト目も'¥0'と見なされ、そこが文字列の終わりであると判断されます。

■例

strcmpとjstrcmpを用いて文字列の比較を行います。返す値が異なるところに注目してください。

```
#include <stdio.h>
#include <string.h>
#include <jstring.h>

char *str1 = "日本語文字列１２３４５６７";
char *str2 = "ﾊﾝｶｸﾉ､ ｶﾅﾓｼﾞﾚﾂﾊ ﾄﾃﾓ ﾖﾐﾆｸｲ｡";

void main(void)
{
        printf("strcmp(str1, str2)  ... %d¥n", strcmp(str1, str2));
        printf("jstrcmp(str1, str2) ... %d¥n", jstrcmp(str1, str2));
}
```

jstrlen

■書式

#include <jstring.h>

int jstrlen (str);

const unsigned char *str; 文字列

■戻り値

文字列strの文字数を返します。

■機能

jstrlen関数は文字列strの文字数を数えます。全角文字も1文字として数えます。

■ポイント

strlen関数の日本語対応版です。全角文字の2バイト目にあたる文字が'¥0'の場合は、その文字の1バイト目も'¥0'であると見なされ、文字数に含めません。

■例

strlen, jstrlen関数を同じ文字列に適用して違いを示します。

```
#include <stdio.h>
#include <string.h>
#include <jstring.h>

main()
{
        unsigned char *ptr = "strlen関数とjstrlen関数";

        printf("文字列      : %s¥n", ptr);
        printf(" strlen関数 : %d¥n", strlen(ptr));
        printf("jstrlen関数 : %d¥n", jstrlen(ptr));
}
```

jstrmatch

■書式

＃include ＜jstring.h＞

char ＊jstrmatch (str1, str2)；

const unsigned char ＊str1；文字列1
const unsigned char ＊str2；文字列2

■戻り値

文字列1へのポインタを返します。詳しくはstrpbrk関数を参照してください。

■機能

jstrmatch関数はstrpbrk関数の日本語対応版です。働きはstrpbrk関数と同じですが、文字列1、2ともに全角文字を使用することができます。

jstradv関数は、文字列2の中のどれか1文字も文字列1に含まれていない場合は、文字列1の最後の'¥0'を指すポインタを返します。NULLポインタを返すのではないことに注意してください。

■ポイント

他の日本語文字列処理関数と同様に、文字列の中の全角文字の2バイト目にあたる文字が'¥0'の場合は、1バイト目も'¥0'であると判断され、そこで文字列が終了しているものと見なされます。

■例

文字列中に句読点があるかどうかを調べます。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr1, *ptr2;
        ptr1 = "jstrmatch関数は、stpbrk関数の日本語対応版です。";
        ptr2 = "、。，．";

        printf("文字列:%s¥n¥n", ptr1);

        if (*jstrmatch(ptr1, ptr2) != '¥0')
                printf("句読点があります。¥n");
        else
                printf("句読点がありません。¥n");
}
```

jstrncat
jstrncmp
jstrncpy

■書式

#include <jstring.h>

unsigned char *jstrncat (str1, str2, n);
int jstrncmp (str1, str2, n);
unsigned char *jstrncpy (str1, str2, n);

const unsigned char *str1; 文字列1
(jstrncpy関数ではunsigned char *str1;)
const unsigned char *str2; 文字列2
size_t n; 文字数

■戻り値

jstrncat, jstrncmp, jstrncpy関数とも、それぞれstrncat, strncmp, strncpy関数と同じ値を返します。

■機能

jstrncat関数はstrncat関数の日本語対応版です。strncat関数ではnはバイト数でしたが、jstrcat関数では〝文字数〟として与えます。

jstrncmp関数はstrncmp関数の日本語対応版です。nは〝文字数〟として与えます。また、jstrncmp関数での大小関係は以下のようになります。

ANK文字<半角カナ文字<全角文字

jstrncpy関数はstrncpy関数の日本語対応版です。やはりnを〝文字数〟として与えます。

■ポイント

全角文字の2バイト目にあたる文字が'¥0'の場合、それぞれの関数は1バイト目も'¥0'と見なします。

それぞれの関数のうち、引数nについてはバイト数ではなく文字数を指定します。全角文字も1文字として数えます。

■例

全角文字を含む文字列の文字数指定付きの連結を行います。

```
#include <stdio.h>
#include <string.h>
#include <jstring.h>

main()
{
        unsigned char *ptr1, *ptr2, buf[128];

        ptr1 = "これは全角文字です";
        ptr2 = "ｺﾚﾊ ﾊﾝｶｸ ｶﾅ ﾓｼﾞ ﾃﾞｽ ";

        printf("文字列１ : %s¥n文字列２ : %s¥n¥n", ptr1, ptr2);

        strncpy(buf, ptr1, 4);
        printf(" strncpy関数:%s¥n", buf);
        jstrncpy(buf, ptr1, 4);
        printf("jstrncpy関数:%s¥n¥n", buf);

        strcpy(buf, ptr2);
        strncat(buf, ptr1, 6);
        printf(" strncat関数:%s¥n", buf);
        strcpy(buf, ptr2);
        jstrncat(buf, ptr1, 6);
        printf("jstrncat関数:%s¥n", buf);
}
```

jstrrchr

■書式

#include <jstring.h>

unsigned char *jstrrchr (str, c) ;

const unsigned char *str ;　文字列
unsigned short c ;　　文字

■戻り値

strrchr関数と同じ値を返します。

■機能

jstrrchr関数はstrrchr関数の日本語対応版です。文字cはstrrchr関数と違い、char型ではなくunsigned short型として与えます。

■ポイント

文字列strの中の全角文字のうち2バイトが'¥0'の文字は1バイト目も'¥0'と判断され、その時点で文字列が終了したと見なされます。

■例

文字列を日本語文字列で検索します。

```
#include <stdio.h>
#include <jstring.h>

unsigned short jcode(unsigned char *c);

main()
{
        unsigned char *ptr;
        ptr = "これは、jstrrchr関数のサンプルです。";

        printf("文字列:%s¥n", ptr);

        if (jstrrchr(ptr, jcode("、")) != NULL)
                printf("読点があります。¥n");
        if (jstrrchr(ptr, jcode("。")) != NULL)
                printf("句点があります。¥n");
}

        /* 文字列の先頭の漢字コードを返す */

unsigned short jcode(unsigned char *p)
{
        return ((*p << 8) | *(p + 1));
}
```

jstrrev

■書式

#include <jstring.h>

unsigned char ＊jstrrev (str)；

unsigned char ＊str；　　文字列

■戻り値

反転した文字列strの先頭を指すポインタを返します。

■機能

jstrrev関数はstrrev関数の日本語対応版です。jstrrev関数では全角文字は1文字として扱われ、文字の1バイト目と2バイト目が反転することはありません。

■ポイント

文字列strそのものを変更してしまうので、元の文字列を使用する場合は先にstrcpy等で複写しておいてください。

■例

回文(上から読んでも下から読んでも同じ文)かどうかをチェックします。

```
#include <stdio.h>
#include <string.h>
#include <jstring.h>

void    kaibun(unsigned char *str);

main()
{
        printf("回文をチェックします。¥n");
        kaibun("たけやぶやけた");
        kaibun("しんかんせんえんせんかんし");
        kaibun("これはちがいます");
}

void    kaibun(unsigned char *str);
{
        unsigned char *ptr;
        ptr = (unsigned char *)strdup(str);
        ptr = jstrrev(ptr);
        printf("%s:", str);
        if (jstrcmp(str, ptr) == 0)
                printf("これは回文です。¥n");
        else
                printf("これは回文ではありません。¥n");
        free(ptr);
}
```

jstrskip

■書式

#include <jstring.h>

unsigned char ＊jstrskip (str1, str2)；

unsigned char ＊str1；文字列1
unsigned char ＊str2；文字列2

■戻り値

文字列2の文字のうち、どれにも当てはまらない文字が最初に文字列1に現れる位置を指すポインタを返します。

文字列1の中の文字がすべて文字列2の中の文字にあてはまる場合は、文字列1の最後の'¥0'を指すポインタを返します。

■機能

jstrskip関数は文字列1の中の文字を左から1文字ずつ調べ、その文字が文字列2の中にひとつもあてはまらない場合は、その位置を指すポインタを返します。

jstrskip関数はstrspn関数と似たような動作をしますが、strspn関数が「その位置までの文字の長さ」を返すのに対し、jstrskip関数は「その位置を指すポインタ」を返します。また、jstrskip関数では文字列1、文字列2ともに全角文字を含むことができ、正しく検索します。

■ポイント

文字列1、2ともに、中に含まれる全角文字の2バイト目にあたる文字が'¥0'の場合、1バイト目も'¥0'であると見なされ、そこで文字列が終了していると判断されてしまいます。

■例

日本語文字列のスペース、タブ、'.'以外の部分の先頭位置を求めます。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr1, *ptr2;
        ptr1 = "              ....jstrskip関数のテストです。";
        ptr2 = " ¥t .";
        /*      ↑全角スペース */
        printf("文字列:%s¥n", ptr1);
        printf("結果:%s¥n", jstrskip(ptr1, ptr2));
}
```

jstrstr

■書式

#include <jstring.h>

unsigned char *jstrstr (str1, str2);

**const unsigned char ***str1; 文字列1
**const unsigned char ***str2; 文字列2

■戻り値

文字列2が文字列1の一部分に現れる場合、その最初の位置を指すポインタを返します。現れない場合はNULLを返します。

詳しくはstrstr関数を参照してください。

■機能

jstrstr関数はstrstr関数の日本語対応版です。働きはstrstr関数と同じですが、文字列の中に全角文字が含まれても正しく動作します。

■ポイント

strstr関数の日本語対応版です。他の関数と同様、全角文字の2バイト目にあたる文字が'¥0'である場合、1バイト目も'¥0'と判断され、そこが文字列の終了と見なされます。

■例

名前を入力すると、入力された名前に応じて答えます。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char buf[128];
        fputs("漢字でお名前をどうぞ:", stdout);
        gets(buf);

        if (jstrstr(buf, "鈴木") != NULL)
                puts("鈴木さん、こんにちはっ!");
        else if (jstrstr(buf, "若田部") != NULL)
                puts("こんにちは、若田部さん。お元気でしたか?");
        else if (jstrstr(buf, "木村") != NULL)
                puts("こんにちは。景気はいかがですか、木村さん?");
        else if (jstrstr(buf, "斎藤") != NULL)
                puts("こんにちは。車の調子はどうですか?");
        else
                puts("はじめまして。");
}
```

jstrtok

■書式

#include <jstring.h>

unsigned char *jstrtok (str1, str2)；

unsigned char *str1；
unsigned char *str2；

■戻り値

トークンが切り出せた場合には、切り出したトークンへのポインタを返します。トークンの切り出しに失敗した場合はNULLを返します。

■機能

str2をデリミタとしてstr1からトークンを切り出します。次のトークンを切り出すためにはstr1をNULLにして呼び出します。str1が日本語文字列でも正常に切り出すことができます。str2のデリミタとして日本語文字列を与えることもできます。

■ポイント

strtok関数の日本語対応版です。

■例

標準入力から入力された文字列の句読点や空白を区切りとして、1行ずつ出力します。

```
#include <stdio.h>
#include <jstring.h>

static char buffer[1024];
char *dlm = "　、。¥t¥n .,";
/*          ↑         ↑半角スペース */
/*      全角スペース               */

void main(void)
{
        char *p;

        while (fgets(buffer, sizeof(buffer), stdin) != NULL) {
                for (p = jstrtok(buffer, dlm);
                     p != NULL; p = jstrtok(NULL, dlm))
                        puts(p);
        }
}
```

jtolower jtohira
jtoupper jtokata

■書式

```
#include <jstring.h>

unsigned short jtolower (c);
unsigned short jtoupper (c);
unsigned short jtohira (c);
unsigned short jtokata (c);

int c;            変換する文字
```

■戻り値

それぞれ、関数によって全角文字を変換し、変換した結果をShift-JISコードで返します。変換できない文字の場合は文字cの値をそのまま返します。

■機能

jtolowerは全角文字cが英大文字だった場合は小文字に変換します。その他の文字については変換しません。

jtoupperは全角文字cが英小文字だった場合は大文字に変換します。その他の文字については変換しません。

jtohiraは全角文字cが全角カタカナだった場合は全角ひらがなに変換します。半角カナ等その他の文字については変換しません。

jtokanaは全角文字cが全角ひらがなだった場合は全角カタカナに変換します。その他の文字については変換しません。

■ポイント

全角文字cが上で示した条件に合わない場合は文字cのコードをそのまま返しますので、jiskana等の関数でチェックする必要はありません。

■例

全角かなをカナに、大文字を小文字に変更します。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr, *prt;
        unsigned short c;
        ptr = "全角かなをカナに、大文字（ＡＢＣ・・）を小文字に"
              "変更します。";
        prt = ptr;

        puts(prt);
        while (*ptr != '¥0') {
                c = (*ptr << 8) | *(ptr + 1);
                c = jtolower(jtokana(c));
                *ptr++ = c >> 8;
                *ptr++ = c & 0xff;
        }
        puts(prt);
}
```

mtob

■書式

```
#include <jstring.h>

int mtob (str, nmoji);

const unsigned char *str; 文字列
int nmoji;                文字数
```

■戻り値

文字列strのnmoji目までのバイト数を返します。

■機能

mtob関数はbtom関数の逆の働きをする関数です。mtob関数は文字列strのnmoji文字目までに何バイトあるかを数えます。普通のキャラクタは1文字で1バイト、全角文字は1文字で2バイトとして数えます。

■ポイント

nmoji文字目より前に'¥0'が現れた場合(つまり文字列strがnmoji文字より短かった場合)は全体のバイト数を返します。

また、全角文字の2バイト目にあたる文字が'¥0'の場合、その文字は無視されます。

■例

1行入力し、10文字目までに何バイトあるかを表示します。

```
#include <stdio.h>
#include <jstring.h>

void    star(int len);

main()
{
        unsigned char buf[128];
        int i, len;

        printf("日本語文を入力して下さい。¥n");
        gets(buf);

        len = mtob(buf, 10);
        if (len == 0)
                printf("何も入力されませんでした。¥n");
        else {
                printf("%dバイト入力されました。¥n", len);
                star(len);
                puts(buf);
                star(len);
        }
}

void    star(int len)
{
        int i;
        for (i = 0; i < len; i++)
                putchar('*');
        putchar('¥n');
}
```

nthctype

■書式

```
#include <jstring.h>

int nthctype (str, nbyte);

unsigned char *str; 文字列
int nbyte;          バイト数
```

■戻り値

文字の種類により、以下の値を返します。

戻り値	意味
CT_ANK	ANK文字
CT_KJ1	漢字の第1バイト目
CT_KJ2	漢字の第2バイト目
CT_ILGL	無効な文字

CT_ANK, CT_KJ1などの記号定数はjctype.hで定義されています。

■機能

nthctype関数はchkctype関数と似た働きをします。nthctype関数では文字列strのnbyteバイト目の文字がどのような種類の文字かを調べます。

chkctype関数のようにtypeを指定しなくても、nthctype関数は、その位置から全角文字の第2バイトとして処理するか、などの判断を行います。

nthctype関数は、以下の場合にはCT_ILGLを返します。

- nbyteに0を指定した。
- 指定された位置の文字が全角文字の第1バイトの直後にあり、かつ、全角文字の第2バイトとして正しくない文字であった。

■ポイント

nthctype関数は指定された位置以外の文字のチェックは行いません。指定された位置以前に'¥0'以外の無効な文字が現れても、それは無視されます。

■例

文字列を入力し、それぞれの要素のJISコード(1バイト、2バイト共)を表示します。

```
#include <stdio.h>
#include <string.h>
#include <jstring.h>
#include <jctype.h>

main()
{
        unsigned char *ptr1;
        int len, i, l;

        ptr1 = "文字列ｺｰﾄﾞ(ascii/shift jis)を表示します。";

        puts(ptr1);
        len = strlen(ptr1);

        for (i = 0; i < len; i++) {
                if (nthctype(ptr1, i) == CT_KJ1) {
                        l = (ptr1[i] << 8) | ptr1[i + 1];
                        printf("%04X ", l);
                        i++;
                }
                else
                        printf("%02X ", ptr1[i]);
        }
}
```

zentohan

■書式

#include <jstring.h>

unsigned short zentohan (c)；

unsigned short c；　　変換する文字

■戻り値

全角文字cを半角文字に変換します。ただし、全角文字cが半角文字に変換できない場合はcがそのまま返されます。

■機能

zentohan関数はhantozen関数の逆の働きをする関数です。

zentohan関数は全角文字cを半角文字に変換します。(例：８→8)

■ポイント

ASCIIコードの0x20から0x7eに対応する全角文字しか変換されません。

■例

全角文字列、"This　is　zentohan's　test."を、半角文字列に変換して出力します。

```
#include <stdio.h>
#include <jstring.h>

main()
{
        unsigned char *ptr;
        unsigned short c;

        ptr = "Ｔｈｉｓ　ｉｓ　ｚｅｎｔｏｈａｎ'ｓ　ｔｅｓｔ．";

        while (*ptr != '¥0') {
                c = (*ptr << 8) | *(ptr + 1);
                c = zentohan(c);
                putchar(c);
        }
}
```

_arc ver 5

■書式

```
#include <graph.h>

short far _arc (x1, y1, x2, y2, x3, y3, x4, y4);

short x1, y1, x2, y2;      座標
short x3, y3, x4, y4;      ベクトル
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

対角が論理座標(x1, y1), (x2, y2)の長方形に内接する楕円の弧を描画します。弧の開始、終了地点は中心からのベクトル(x3, y3), (x4, y4)で指定します。

■ポイント

真円の弧もこの関数で描画します。

_clearscreen　ver 5

■書式

#include <graph.h>

void far _clearscreen (area);

short area;　　領域

■戻り値

ありません。

■機能

areaで指定される条件で画面を消去します。areaにはgraph.hで定義される以下の数値を設定します。

_GCLEARSCREEN	グラフィック・テキストを消去します
_GCLEARGRAPH	グラフィックを消去します
_GCLEARTEXT	テキストを消去します
_GVIEWPORT	ビューポート内を消去します
_GWINDOW	ウィンドウ内を消去します

_displaycursor　ver 5

■書式

```
#include <graph.h>

short far _displaycursor (flag);

short flag;        カーソルの状態
```

■戻り値

直前の設定値を返します。

■機能

テキストカーソルの表示・非表示を切りかえます。flagには以下の数値を指定します。

_GCURSOROFF	非表示
_GCURSORON	表示

_ellipse　　ver 5

■書式

#include <graph.h>

short far _ellipse (mode, x1, y1, x2, y2)；

short mode；	フィルモード
short x1, y1；	左上角の座標
short x2, y2；	右下角の座標

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

対角が論理座標(x1, y1), (x2, y2)の長方形に内接する楕円を描画します。modeには次のような定数値を与えます。

_GFILLINTERIOR	カレントフィルマスクで楕円の内部を塗りつぶします。
_GBORDER	楕円の枠のみを描画します。

■ポイント

真円もこの関数で描画します。

_floodfill ver 5

■書式

```
#include <graph.h>

short far _floodfill (x, y, boundary);

short x, y;          開始位置
short boundary;      境界線の色
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

論理座標(x, y)から、境界色boundaryで囲まれる領域を塗りつぶします。

_getbkcolor ver 5

■書式

```
#include <graph.h>

long far _getbkcolor (void);
```

■戻り値

エラー戻り値はありません。

■機能

現在のバックグラウンド色を取得します。

_getcolor　ver 5

■書式

```
#include <graph.h>

short far _getcolor (void);
```

■戻り値

エラー戻り値はありません。

■機能

カレントカラーを取得します。

_getcurrentposition　ver 5

■書式

```
#include <graph.h>

struct xycoord far _getcurrentposition (void);
```

■戻り値

エラー戻り値はありません。

■機能

カレントポジションを構造体xycoordで返します。

■ポイント

カレントポジションは _arc, _lineto, _movetoで移動します。

_getfillmask ver 5

■書式
```
#include <graph.h>

unsigned char far * far _getfillmask (mask);

unsigned char far *mask;        マスク配列
```

■戻り値

カレントフィルマスクが設定されていなければNULLを返します。

■機能

カレントフィルマスクをmaskに取得します。maskはunsigned characterへのfarポインタです。

■ポイント

_floodfillや、_pie, _ellipse, _rectangleで領域内の塗りつぶしを指定した場合にはカレントフィルマスクにより塗りつぶしが行われます。

_getimage ver 5

■書式
```
#include <graph.h>

void far _getimage (x1, y1, x2, y2, buf);

short x1, y1;                左上角の座標
short x2, y2;                右下角の座標
unsigned char far *buf;      バッファ
```

■戻り値

ありません。

■機能

グラフィック画面の論理座標(x1, y1), (x2, y2)を対角とする領域のデータをbufに転送します。bufはunsigned charへのfarポインタです。

_getkanji ver 5

■書式

```
#include <graph.h>

short far _getkanji (jis, buf);

unsinged int jis;              JIS漢字コード
unsigned char far *buf;        文字パターン配列
```

■戻り値

成功した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

JISコードjisの漢字フォントをbufに取得します。bufはunsigned charへのfarポインタです。

_getlinestyle ver 5

■書式

```
#include <graph.h>

unsigned short _getlinestyle (void);
```

■戻り値

エラー戻り値はありません。

■機能

カレントラインスタイルを取得します。

■ポイント

グラフィック画面に線を描画する関数はラインスタイルとしてカレントラインスタイルを使用します。

_getlogcoord　ver 5

■書式
#include <graph.h>

```
struct xycoord far _getlogcoord (x, y);

short x;    物理座標
short y;    物理座標
```

■戻り値
エラーを表わす戻り値はありません。

■機能
物理座標(x, y)を論理座標に変換して、構造体xycoord型で返します。xycoordはgraph.hで定義されています。

_getphyscoord　ver 5

■書式
#include <graph.h>

```
struct xycoord far _getphyscoord (x, y);

short x;    論理座標
short y;    論理座標
```

■戻り値
エラーを表わす戻り値はありません。

■機能
論理座標(x, y)を物理座標に変換して、構造体xycoord型で返します。xycoordはgraph.hで定義されています。

_getpixel　　ver 5

■書式

```
#include <graph.h>

short far _getpixel (x, y);

short x, y;        ピクセルの座標
```

■戻り値

エラーが発生した場合は-1を返します。

■機能

論理座標(x, y)に表示されているピクセル値を返します。

_gettextcolor　　ver 5

■書式

```
#include <graph.h>

short far _gettextcolor (void);
```

■戻り値

エラー戻り値はありません。

■機能

現在のテキスト表示の文字色を取得します。

_gettextposition　ver 5

■書式

```
#include <graph.h>

struct rccoord far _gettextposition (void);
```

■戻り値

エラー戻り値はありません。

■機能

現在のテキストカーソル位置を返します。xycoordはgraph.hで定義されています。

■ポイント

標準入出力関数やコンソール入出力関数には影響されません。

_getvideoconfig　ver 5

■書式

#include <graph.h>

struct videoconfig far ＊ far _getvideoconfig (config)；

struct videoconfig far ＊config；　構造体へのポインタ

■戻り値

ありません。

■機能

現在使用できるグラフィック・テキストの状態をconfigに返します。configは構造体videoconfigへのfarポインタであり、構造体videoconfigはgraph.hで次のように定義されています。

```
struct videoconfig {
        short numxpixels；       x方向のピクセル数
        short numypixels；       y方向のピクセル数
        short numtextcols；      テキストの桁数
        short numtextrows；      テキストの行数
        short numcolors；        使用できる色数
        short bitsperpixel；     1ピクセルのドット数
        short numvideopages；    画面ページ数
        short mode；             現在のモード
        short adapter；          アダプタ
        short monitor；          モニタ種別
        short memory；           グラフィックのメモリサイズ
}；
```

_gputchar　　ver 5

■書式

#include <graph.h>

void far cdecl _gputchar (x, y, code, action)；

short x, y；　　左上角の座標
short code；　　文字コード
short action；　　アクション

■戻り値

ありません。

■機能

論理座標(x, y)に文字codeを描画します。actionは描画を行う領域と文字フォントとの作用で次の値を設定します。

_GAND	ANDを取ります	(&)
_GOR	ORをとります	(｜)
_GXOR	XORをとります	(^)
_GPRESET	リバース表示	
_GPSET	ノーマル表示	

_imagesize　　ver 5

■書式

```
#include <graph.h>

long far cdecl _imagesize (x1, y1, x2, y2);

short x1, y1;    左上角の座標
short x2, y2;    右下角の座標
```

■戻り値

エラーを表わす戻り値はありません。

■機能

(x1, y1)-(x2, y2)の領域を_getimage関数で転送するとき、必要となるバッファの容量を取得します。

_lineto ver 5

■書式

```
#include <graph.h>

short far _lineto (x, y);

short x;	終点のX座標
short y;	終点のY座標
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

カレントポジションから論理座標(x, y)までカレントカラー、カレントラインスタイルで線を描画します。

_moveto ver 5

■書式

```
#include <graph.h>

struct xycoord far _moveto (x, y);

short x;	移動先のX座標
short y;	移動先のY座標
```

■戻り値

直前の設定値を返します。

■機能

カレントポジションを論理座標(x, y)に移動します。xycoordはgraph.hで定義されています。

_outtext _outtextg

ver 5

■書式

#include <graph.h>

```
void far cdecl _outtext (text);
void far cdecl _outtextg (text);

char for  * text;        出力するテキスト
```

■戻り値

ありません。

■機能

ウィンドウ内に文字列を表示します。_outtext関数は日本語文字列を表示することができます。_outtextg関数ではシフトJIS第1バイト目に相当するキャラクタは、システムのグラフィックキャラクタとして表示され、漢字表示はできません。

_pie　　ver 5

■書式

```
#include <graph.h>

short far _pie (mode, x1, y1, x2, y2, x3, y3, x4, y4);

short mode;                フィルモード
short x1, y1, x2, y2;      座標
short x3, y3, x4, y4;      ベクトル
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

対角が論理座標(x1, y1), (x2, y2)の長方形に内接する楕円の弧と中心からの線分からなる扇型を描画します。弧の開始、終了地点は中心からのベクトル(x3, y3), (x4, y4)で指定します。modeには次のような定数値を与えます。

_GFILLINTERIOR	カレントフィルマスクで扇型の内部を塗りつぶします
_GBORDER	扇型の枠のみを描画します

■ポイント

真円の扇型もこの関数で描画します。

_putimage　　ver 5

■書式

```
#include <graph.h>

void far _putimage (x, y, image, action);

short x, y;             左上角の座標
char far *image;        バッファ
short action;           アクション
```

■戻り値

ありません。

■機能

_getimageで取得したグラフィックデータimageを論理座標(x, y)に描画します。actionは描画を行う領域とグラフィックデータとの作用で次の値を設定します。

_GAND	ANDをとります	(&)
_GOR	ORをとります	(|)
_GXOR	XORをとります	(^)
_GPRESET	リバース表示	
_GPSET	ノーマル表示	

_rectangle　ver 5

■書式

```
#include <graph.h>

short far _rectangle (mode, x1, y1, x2, y2);

short mode;        フィルモード
short x1, y1;      左上角の座標
short y2, y2;      右下角の座標
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

論理座標(x1, y1)-(x2, y2)を対角とする長方形をカレントカラー、カレントラインスタイルで描画します。modeには次のような定数値を与えます。

_GFILLINTERIOR　カレントフィルマスクで楕円の内部を塗りつぶします
_GBORDER　楕円の枠のみを描画します

_remapallpalette　ver 5

■書式

#include <graph.h>

short far _remapallpalette (colors);

long far *colors;　　カラー番号の配列

■戻り値

変更できた場合には0を返します。変更できなかった場合は－1Lを返します。

■機能

パレット配列をすべて変更します。colorsはパレット値の配列で、配列の要素は現在のビデオモードでの最大表示色分必要です。

_remappalette　ver 5

■書式

#include <graph.h>

long far _remappalette (color, palette);

short color;　　カラー番号
long palette;　　カラー番号に対応するピクセル値

■戻り値

変更できた場合には、変更前のパレットに対応した色コードを返します。変更できなかった場合には－1Lを返します。

■機能

colorに対応する発光色をpaletteに変更します。

_selectpalette　ver 5 ［AX］

■書式

#include <graph.h>

short far _selectpalette (number)；

short number；　　カラー番号

■戻り値

変更前の設定値を返します。エラー戻り値はありません。

■機能

パレットを変更します。AXシリーズのビデオモードMRES4COLORとMRESNOCOLORにおいてのみ有効で、スクリーンモード・ビデオオプションに応じてnumberには以下の値を設定します。

［MRES4COLOR］

値	COLOR1	COLOR2	COLOR3
0	緑	赤	紫
1	シアン	マゼンタ	淡い灰色
2	淡い緑	淡い赤	黄
3	淡いシアン	淡いマゼンタ	白

［MRESNOCOLOR］　CGAを使用している場合

値	COLOR1	COLOR2	COLOR3
0	緑	赤	淡い灰色
1	淡い青	淡い赤	白

［MRESNOCOLOR］　JEGAを使用している場合

値	COLOR1	COLOR2	COLOR3
0	緑	赤	茶
1	淡い緑	淡い赤	黄
2	淡いシアン	淡い赤	黄

■ポイント

9801シリーズでは使用できません。

_setactivepage　　ver 5

■書式

```
#include <graph.h>

short far _setactivepage (page);

short page;          ページ番号
```

■戻り値

直前の設定値を返します。エラーが発生すると負の値を返します。

■機能

描画を行うグラフィックページを設定します。ハードウェアによって設定できる範囲が違います。

旧9801-U2では0のみ、それ以外の9801では0または1が設定できます。AXシリーズの場合、実装されているオプションやモードにより変化します。

■ポイント

表示を行う画面を設定するには、_setvisualpage()関数を使用してください。

_setbkcolor　　ver 5

■書式

#include <graph.h>

long far _setbkcolor (pixel);

long pixel;　　　カラー番号

■戻り値

直前のバックグラウンドピクセル値を返します。

■機能

バックグラウンド色をpixelに変更します。

■ポイント

remappalette(0, pixel)と同義です。

_setcliprgn　　ver 5

■書式

```
#include <graph.h>

void far _setcliprgn (x1, y1, x2, y2);

int x1, y1;      左上角の座標
int x2, y2;      右下角の座標
```

■戻り値

ありません。

■機能

クリップ領域を物理座標(x1, y1) - (x2, y2)に設定します。

■ポイント

クリップ領域とは描画を行う範囲で、クリップ領域を設定すると設定した範囲外へのグラフィック描画は行われなくなります。

_setcolor　ver 5

■書式

#include <graph.h>

short far _setcolor（color）;

short color；　　カラー番号

■戻り値

直前の設定値を返します。

■機能

カレントカラーをcolorに変更します。

■ポイント

グラフィック描画関数はカレントカラーで描画を行います。_setcolorは描画色変更を行う関数です。

_setfillmask　ver 5

■書式

#include <graph.h>

void far _setfillmask（mask）;

unsigned char far ・mask；　　マスク配列

■戻り値

ありません。

■機能

カレントフィルマスクをmaskに設定します。

■ポイント

_floodfillや _pie, _ellipse, _rectangleで領域内の塗りつぶしを指定した場合にはカレントフィルマスクにより塗りつぶしが行われます。

_setkanji　　ver 5

■書式

```
#include <graph.h>

short far _setkanji (jis, font);

unsigned short code;            文字コード
unsinged char far *font;        文字パターン配列
```

■戻り値

成功した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

JISコードcodeに相当する外字フォントイメージをfontにします。fontはunsigned char型のfarポインタです。

_setlinestyle　　ver 5

■書式

```
#include <graph.h>

void far _setlinestyle (mask);      ラインスタイル・マスク

unsigned short mask;
```

■戻り値

戻り値はありません。

■機能

カレントラインスタイルmaskにします。

■ポイント

グラフィック画面に線を描画する関数はラインスタイルとしてカレントラインスタイルを使用します。ラインスタイルを設定するとにより破線や点線を描画することができます。

_setlogorg　　ver 5

■書式

#include <graph.h>

struct xycoord far _setlogorg (x, y);

short x, y;　　新しい原点の座標

■戻り値

直前の論理座標原点を返します。エラーを表わす戻り値はありません。

■機能

論理座標系の原点を、物理座標(x, y)に設定します。

■ポイント

戻り値は構造体へのポインタではなく構造体そのものですから注意してください。xycoordはgraph.hで定義されています。

_setpixel　　ver 5

■書式

```
#include <graph.h>

short far _setpixel (x, y);

short x, y;        目的のピクセル
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

論理座標(x, y)にピクセルをセットます。

■ポイント

表示される色はカレントカラーとなります。

_settextcolor　ver 5

■書式

```
#include <graph.h>

short far _settextcolor (color);

short color;        カラー番号
```

■戻り値

直前の設定値を返します。エラーが発生すると負の値を返します。

■機能

テキスト表示の文字色をcolorにします。

_settextposition　ver 5

■書式

```
#include <graph.h>

struct rccoord far _settextposition (y, x);

short x, y;        テキスト座標
```

■戻り値

直前のテキストポジションを返します。

■機能

テキスト表示を行う位置を設定します。

■ポイント

返す値は構造体へのポインタではなく、そのものなので注意してください。構造体rccoordはgraph.hで定義されています。

_settextwindow ver 5

■書式

```
#include <graph.h>

void far _settextwindow (r1, c1, r2, c2);

short r1, c1;       左上角の座標
short r2, c2;       右下角の座標
```

■戻り値

ありません。

■機能

テキスト表示を行うウィンドウをテキスト座標(r1, c1)-(r2, c2)を対角とする領域に設定します。

■ポイント

ウィンドウに対して出力を行う場合、_outtext関数を用いる必要があります。標準ストリーム関数やコンソール出力関数により出力を行う場合には、ウィンドウ領域に無関係に出力を行います。

_setvideomode ver 5

■書式

```
#include <graph.h>

short far _setvideomode (mode);

short mode;       モード
```

modeの値	意　味	
_DEFAULTMODE	デフォルトモード (_98TEXT80と同じ)	
_98TEXT80	80×25	デジタル8色(グラフィック表示なし)
_98RESSCOLOR	640×400	デジタル8色
_98RESS8COLOR	640×400	アナログ8色
_98RESS16COLOR	640×400	アナログ16色
_98RESCOLOR	640×400	デジタル8色(テキスト表示なし)
_98RES8COLOR	640×400	アナログ8色(テキスト表示なし)
_98RES16COLOR	640×400	アナログ16色(テキスト表示なし)

■戻り値

成功した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

スクリーンモードをmodeに変更します。modeには上表の定数を指定します。

_setviewport　ver 5

■書式
#include <graph.h>

void far _setviewport (x1, y1, x2, y2)；

int x1, y1;　　左上角の座標
int x2, y2;　　右下角の座標

■戻り値
ありません。

■機能
ビューポートを論理座標(x1, y1) - (x2, y2)に設定します。

■ポイント
クリップ領域を物理座標(x1, y1) - (x2, y2)に設定し、論理座標原点(x1, y1)に設定することと同様の効果を持ちます。

_setvisualpage ver 5

■書式

```
#include <graph.h>

short far _setvisualpage (page);

short page;        ビジュアルページ番号
```

■戻り値

直前の設定値を返します。エラーが発生すると負の値を返します。

■機能

画面表示を行うグラフィックページ番号をpageにします。

■ポイント

描画を行う画面を設定するには_setactive()関数を使用してください。

_tilepaint　ver 5

■書式

```
#include <graph.h>

short far _tilepaint (x, y, boundary, tile);

short x, y;                         開始位置
short boundary;                     境界線の色
unsigned char far *tile;            タイルパターン配列
```

■戻り値

正常に終了した場合は0以外の値を返します。エラーが発生した場合は0を返します。

■機能

論理座標(x, y)から、境界色boundaryで囲まれる領域をtileで塗りつぶします。

■ポイント

tileはnバイトで構成され(16色モードのときn == 4、8色モードのとき n == 3)、BRGの順でタイルパターンを指定します。

_wrapon　ver 5

■書式

#include <graph.h>

short far cdecl _wrapon (option);

short option;　　ラップ条件

■戻り値

直前の設定値を返します。

■機能

テキストがウィンドウ右端を越えて表示されようとするときの処理を選択します。optionには次の値を設定します。

_GWRAPOFF	ウィンドウ範囲を越える部分は表示しません
_GWRAPON	ウィンドウ範囲を越える部分は次の行に表示します

【イージーリファレンス】

このイージーリファレンスはある程度C言語がわかっている人を対象に作られた、関数名、戻り値の型、引数の型についてのみの表です。インクルードファイルごとにまとめ、アルファベット順としてあります。各関数の動作や引数の意味についてはリファレンス本文で詳しく説明してあります。

CONIO.H

char *cgets(char *)
int cprintf(char *, ...)
int cputs(char *)
int cscanf(char *, ...)
int getch(void)
int getche(void)
int inp(unsigned int)
unsigned inpw(unsigned int)
int kbhit(void)
int outp(unsigned int, int)
unsigned outpw(unsigned int, unsigned int)
int putch(int)
int ungetch(int)

CTYPE.H

int isalpha(int)
int isupper(int)
int islower(int)
int isdigit(int)
int isxdigit(int)
int isspace(int)
int ispunct(int)
int isalnum(int)
int isprint(int)
int isgraph(int)
int iscntrl(int)
int isascii(int)
int _tolower(int)
int _toupper(int)
int toupper(int)
int tolower(int)
int toascii(int)
int iscsymf(int)
int iscsym(int)

DIRECT.H

int chdir(char *)

```
char *getcwd(char *, int)
int mkdir(char *)
int rmdir(char *)
```

DOS.H

```
unsigned FP_SEG(void far *)
unsigned FP_OFF(void far *)
int bdos(int, unsigned int, unsigned int)
void _disable(void)
unsigned _dos_allocmem(unsigned, unsigned *)
unsigned _dos_close(int)
unsigned _dos_creat(char *, unsigned, int *)
unsigned _dos_creatnew(char *, unsigned, int *)
unsigned _dos_findfirst(char *, unsigned, struct find_t *)
unsigned _dos_findnext(struct find_t *)
unsigned _dos_freemem(unsigned)
void _dos_getdate(struct dosdate_t *)
void _dos_getdrive(unsigned *)
unsigned _dos_getdiskfree(unsigned, struct diskfree_t *)
unsigned _dos_getfileattr(char *, unsigned *)
unsigned _dos_getftime(int, unsigned *, unsigned *)
void _dos_gettime(struct dostime_t *)
void _dos_keep(unsigned, unsigned)
unsigned _dos_open(char *, unsigned, int *)
unsigned _dos_setblock(unsigned, unsigned, unsigned *)
unsigned _dos_setdate(struct dosdate_t *)
void _dos_setdrive(unsigned, unsigned *)
unsigned _dos_setfileattr(char *, unsigned)
unsigned _dos_setftime(int, unsigned, unsigned)
unsigned _dos_settime(struct dostime_t *)
int dosexterr(struct DOSERROR *)
void _enable(void)
void _hardresume(int)
void _hardretn(int)
int intdos(union REGS *, union REGS *)
int intdosx(union REGS *, union REGS *, struct SREGS *)
int int86(int, union REGS *, union REGS *)
int int86x(int, union REGS *, union REGS *, struct SREGS *)
void segread(struct SREGS *)
void _chain_intr(void (interrupt far *)())
void (interrupt far *_dos_getvect(unsigned))()
unsigned _dos_read(int, void far *, unsigned, unsigned *)
void _dos_setvect(unsigned, void (interrupt far *)())
unsigned _dos_write(int, void far *, unsigned, unsigned *)
void _harderr(void (far *)())
```

FLOAT.H

unsigned int _clear87(void)
unsigned int _control87(unsigned int,unsigned int)
void _fpreset(void)
unsigned int _status87(void)

IO.H

int access(char *, int)
int chmod(char *, int)
int chsize(int, long)
int close(int)
int creat(char *, int)
int dup(int)
int dup2(int, int)
int eof(int)
long filelength(int)
int isatty(int)
int locking(int, int, long)
long lseek(int, long, int)
char *mktemp(char *)
int open(char *, int, ...)
int read(int, char *, unsigned int)
int remove(char *)
int rename(char *, char *)
int setmode(int, int)
int sopen(char *, int, int, ...)
long tell(int)
int umask(int)
int unlink(char *)
int write(int, char *, unsigned int)

JCTYPE.H

int iskana(int)
int iskpun(int)
int iskmoji(int)
int isalkana(int)
int ispnkana(int)
int isalnmkana(int)
int isprkana(int)
int isgrkana(int)
int iskanji(int)
int iskanji2(int)

JSTRING.H

```
int btom(unsigned char *, int)
int chkctype(unsigned char, int)
unsigned short hantozen(unsigned short)
int jisalpha(unsigned short)
int jisdigit(unsigned short)
int jishira(unsigned short)
int jiskata(unsigned short)
int jiskigou(unsigned short)
int jisl0(unsigned short)
int jisl1(unsigned short)
int jisl2(unsigned short)
int jislower(unsigned short)
int jisspace(unsigned short)
unsigned short jistojms(unsigned short)
int jisupper(unsigned short)
int jiszen(unsigned short)
unsigned short jmstojis(unsigned short)
unsigned char *jstradv(unsigned char *, int)
unsigned char *jstrchr(unsigned char *, unsigned short)
int jstrcmp(unsigned char *, unsigned char *)
unsigned int jstrlen(unsigned char *)
unsigned char *jstrmatch(unsigned char *, unsigned char *)
unsigned char *jstrncat(unsigned char *, unsigned char *, unsigned int)
int jstrncmp(unsigned char *, unsigned char *, unsigned int)
unsigned char *jstrncpy(unsigned char *, unsigned char *, unsigned int)
unsigned char *jstrrchr(unsigned char *, unsigned short)
unsigned char *jstrrev(unsigned char *)
unsigned char *jstrskip(unsigned char *, unsigned char *)
unsigned char *jstrstr(unsigned char *, unsigned char *)
unsigned char *jstrtok(unsigned char *, unsigned char *)
unsigned short jtohira(unsigned short)
unsigned short jtokata(unsigned short)
unsigned short jtolower(unsigned short)
unsigned short jtoupper(unsigned short)
int mtob(unsigned char *, int)
int nthctype(unsigned char *, int)
unsigned short zentohan(unsigned short)
```

MALLOC.H

```
void *alloca(unsigned int)
void *calloc(unsigned int, unsigned int)
void *_expand(void *, unsigned int)
int _heapchk(void)
int _fheapchk(void)
int _nheapchk(void)
int _heapset(unsigned int)
int _fheapset(unsigned int)
int _nheapset(unsigned int)
int _heapwalk(struct _heapinfo *)
int _nheapwalk(struct _heapinfo near *)
int _fheapwalk(struct _heapinfo far *)
unsigned int _freect(unsigned int)
void free(void *)
void *malloc(unsigned int)
unsigned int _memavl(void)
unsigned int _memmax(void)
unsigned int _msize(void *)
void *realloc(void *, unsigned int)
void *sbrk(int)
unsigned int stackavail(void)
void _ffree(void far *)
void far *_fmalloc(unsigned int)
unsigned int _fmsize(void far *)
void huge *halloc(long, unsigned int)
void hfree(void huge *)
void _nfree(void near *)
void near *_nmalloc(unsigned int)
unsigned int _nmsize(void near *)
```

MATH.H

```
int abs(int)
double acos(double)
double asin(double)
double atan(double)
double atan2(double, double)
double atof(char *)
double cabs(struct complex)
double ceil(double)
double cos(double)
double cosh(double)
int dieeetomsbin(double *, double *)
int dmsbintoieee(double *, double *)
double exp(double)
double fabs(double)
int fieeetomsbin(float *, float *)
double floor(double)
double fmod(double, double)
int fmsbintoieee(float *, float *)
double frexp(double, int *)
double hypot(double, double)
double j0(double)
double j1(double)
double jn(int, double)
long labs(long)
double ldexp(double, int)
double log(double)
double log10(double)
int matherr(struct exception *)
double modf(double, double *)
double pow(double, double)
double sin(double)
double sinh(double)
double sqrt(double)
double tan(double)
double tanh(double)
double y0(double)
double y1(double)
double yn(int, double)
```

MEMORY.H

```
void *memccpy(void *, void *, int, unsigned int)
void *memchr(void *, int, unsigned int)
int memcmp(void *, void *, unsigned int)
void *memcpy(void *, void *, unsigned int)
int memicmp(void *, void *, unsigned int)
void *memset(void *, int, unsigned int)
void movedata(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int)
```

PROCESS.H

```
void abort(void)
int execl(char *, char *, ...)
int execle(char *, char *, ...)
int execlp(char *, char *, ...)
int execlpe(char *, char *, ...)
int execv(char *, char **)
int execve(char *, char **, char **)
int execvp(char *, char **)
int execvpe(char *, char **, char **)
void exit(int)
void _exit(int)
int getpid(void)
int spawnl(int, char *, char *, ...)
int spawnle(int, char *, char *, ...)
int spawnlp(int, char *, char *, ...)
int spawnlpe(int, char *, char *, ...)
int spawnv(int, char *, char **)
int spawnve(int, char *, char **, char **)
int spawnvp(int, char *, char **)
int spawnvpe(int, char *, char **, char **)
int system(char *)
```

SEARCH.H

```
char *lsearch(char *, char *, unsigned int *, unsigned int, int (*)(void *, void *))
char *lfind(char *, char *, unsigned int *, unsigned int, int (*)(void *, void *))
void *bsearch(void *, void *, unsigned int, unsigned int, int (*)(void *, void *))
void qsort(void *, unsigned int, unsigned int, int (*)(void *, void *))
```

SETJMP.H

```
int setjmp(jmp_buf)
void longjmp(jmp_buf, int)
void (*signal(int, void (*)()))()
```

SIGNAL.H

```
int raise(int)
```

STDIO.H

```
int _filbuf(FILE *)
int _flsbuf(int, FILE *)
void clearerr(FILE *)
int fclose(FILE *)
int fcloseall(void)
FILE *fdopen(int, char *)
int fflush(FILE *)
int fgetc(FILE *)
int fgetchar(void)
int fgetpos(FILE *, long *)
char *fgets(char *, int, FILE *)
int flushall(void)
FILE *fopen(char *, char *)
int fprintf(FILE *, char *, ...)
int fputc(int, FILE *)
int fputchar(int)
int fputs(char *, FILE *)
unsigned int fread(void *, unsigned int, unsigned int, FILE *)
FILE *freopen(char *, char *, FILE *)
int fscanf(FILE *, char *, ...)
int fsetpos(FILE *, long *)
int fseek(FILE *, long, int)
long ftell(FILE *)
unsigned int fwrite(void *, unsigned int, unsigned int, FILE *)
char *gets(char *)
int getw(FILE *)
void perror(char *)
int printf(char *, ...)
int puts(char *)
int putw(int, FILE *)
```

```
int remove(char *)
int rename(char *, char *)
void rewind(FILE *)
int rmtmp(void)
int scanf(char *, ...)
void setbuf(FILE *, char *)
int setvbuf(FILE *, char *, int, unsigned int)
int sprintf(char *, char *, ...)
int sscanf(char *, char *, ...)
char *tempnam(char *, char *)
FILE *tmpfile(void)
char *tmpnam(char *)
int ungetc(int, FILE *)
int unlink(char *)
int vfprintf(FILE *, char *, va_list)
int vprintf(char *, va_list)
int vsprintf(char *, char *, va_list)
int getc(FILE *)
int putc(int, FILE *)
int getchar(void)
int putchar(int)
int feof(FILE *)
int ferror(FILE *)
int fileno(FILE *)
```

STDLIB.H

```
void abort(void)
int abs(int)
type max(type, type)
type min(type, type)
int atexit(void (*)(void))
double atof(char *)
int atoi(char *)
long atol(char *)
void *bsearch(void *, void *, unsigned int, unsigned int, int (*)(void *, void *))
void *calloc(unsigned int, unsigned int)
div_t div(int, int)
char *ecvt(double, int, int *, int *)
void exit(int)
void _exit(int)
char *fcvt(double, int, int *, int *)
void free(void *)
char *gcvt(double, int, char *)
char *getenv(char *)
char *itoa(int, char *, int)
```

```
long labs(long)
ldiv_t ldiv(long, long)
unsigned long _lrotl(unsigned long, int)
unsigned long _lrotr(unsigned long, int)
char *ltoa(long, char *, int)
void _makepath(char *, char *, char *, char *, char *)
void *malloc(unsigned int)
onexit_t onexit(onexit_t)
void perror(char *)
int putenv(char *)
void qsort(void *, unsigned int, unsigned int, int (*)(void *, void *))
unsigned int _rotl(unsigned int, int)
unsigned int _rotr(unsigned int, int)
int rand(void)
void *realloc(void *, unsigned int)
void _searchenv(char *, char *, char *)
void _splitpath(char *, char *, char *, char *, char *)
void srand(unsigned int)
double strtod(char *, char **)
long strtol(char *, char **, int)
unsigned long strtoul(char *, char **, int)
void swab(char *, char *, int)
int system(char *)
char *ultoa(unsigned long, char *, int)
int tolower(int)
int toupper(int)
void *memccpy(void *, void *, int, unsigned int)
```

STRING.H

```
void *memchr(void *, int, unsigned int)
int memcmp(void *, void *, unsigned int)
int memicmp(void *, void *, unsigned int)
void *memcpy(void *, void *, unsigned int)
void *memmove(void *, void *, unsigned int)
void *memset(void *, int, unsigned int)
void movedata(unsigned int, unsigned int, unsigned int, unsigned int, unsigned int)
char *strcat(char *, char *)
char *strchr(char *, int)
int strcmp(char *, char *)
int strcmpi(char *, char *)
int stricmp(char *, char *)
char *strcpy(char *, char *)
unsigned int strcspn(char *, char *)
char *strdup(char *)
char *_strerror(char *)
```

char *strerror(int)
unsigned int strlen(char *)
char *strlwr(char *)
char *strncat(char *, char *, unsigned int)
int strncmp(char *, char *, unsigned int)
int strnicmp(char *, char *, unsigned int)
char *strncpy(char *, char *, unsigned int)
char *strnset(char *, int, unsigned int)
char *strpbrk(char *, char *)
char *strrchr(char *, int)
char *strrev(char *)
char *strset(char *, int)
unsigned int strspn(char *, char *)
char *strstr(char *, char *)
char *strtok(char *, char *)
char *strupr(char *)

TIME.H

char *asctime(struct tm *)
char *ctime(long *)
clock_t clock(void)
double difftime(long, long)
struct tm *gmtime(long *)
struct tm *localtime(long *)
longmktime(struct tm *)
char *_strdate(char *)
char *_strtime(char *)
longtime(long *)
void tzset(void)

GRAPH.H

short _setvideomode(short)
short _setactivepage(short)
short _setvisualpage(short)
struct videoconfig *_getvideoconfig(struct videoconfig *)
struct xycoord _setlogorg(short, short)
struct xycoord _getlogcoord(short, short)
struct xycoord _getphyscoord(short, short)
void _setcliprgn(short, short, short, short)
void _setviewport(short, short, short, short)
void _clearscreen(short)
struct xycoord _moveto(short, short)
struct xycoord _getcurrentposition(void)
short _lineto(short, short)
short _rectangle(short, short, short, short, short)

```
short _ellipse(short, short, short, short, short)
short _arc(short, short, short, short, short, short, short, short)
short _pie(short, short, short, short, short, short, short, short, short)
short _setpixel(short, short)
short _getpixel(short, short)
short _floodfill(short, short, short)
short _tilepaint(short, short, short, unsigned char *)
short _setcolor(short)
short _getcolor(void)
void _setlinestyle(unsigned short)
unsigned short _getlinestyle(void)
void _setfillmask(unsigned char *)
unsigned char *_getfillmask(unsigned char *)
long _setbkcolor(long)
long _getbkcolor(void)
long _remappalette(short, long)
short _remapallpalette(long *)
short _selectpalette(short)
void _settextwindow(short, short, short, short)
void _outtext(char *)
void _outtextg(char *)
short _wrapon(short)
short _displaycursor(short)
struct rccoord _settextposition(short, short)
struct rccoord _gettextposition(void)
short _settextcolor(short)
short _gettextcolor(void)
void _getimage(short, short, short, short, char *)
void _putimage(short, short, char *, short)
long _imagesize(short, short, short, short)
short _getkanji(unsigned short, unsigned char *)
short _setkanji(unsigned short, unsigned char *)
void _gputchar(short, short, unsigned short, short)
```

SYS/STAT.H

```
int fstat(int, struct stat *)
int stat(char *, struct stat *)
```

SYS/TIMEB.H

```
void ftime(struct timeb *)
```

SYS/UTIME.H

```
int utime(char *, struct utimbuf *)
```

【類似関数の機能の違い】

MS-Cで用意されている数多くの関数の中には、同じ機能や似たような機能を持つものがあります。それらは、互換性のために用意されたまったく同じ機能の関数であったり、似てはいるが微妙に違う関数であったり、あるいは、特定の条件では異なる動作をする関数であったりします。ここでは、そのような関数のうち、使用頻度の高いものについて解説します。

★ストリーム／コンソール入出力に関するもの

●文字入力

fgetchar	stdinから1文字読み込む。
getchar (macro)	stdinから1文字読み込む。
fgetc	ストリームから1文字読み込む。
getc (macro)	ストリームから1文字読み込む。
getch	コンソールから1文字読み込む(エコーバックなし)。
getche	コンソールから1文字読み込む(エコーバックあり)。

これらの関数は文字をストリーム／コンソールから読み込みます。fgetcharとgetcharは同機能です。getcとfgetcも同機能となります。また、fgetc(stdin), getc(stdin)はgetchar(), fgetchar()と同機能になります。これらの関数は入力のときにバファリングされますが、getch, getche関数はコンソールからバファリングせずに1文字の読み込みを行います。getch関数はコンソールから入力された文字を画面にエコーバックしませんが、getche関数は入力された文字のエコーバックを行います。また、getch, getche関数以外では、行末文字の変換が行われます。'¥r ¥n'が入力されると、¥nのみが入力データとして読み込まれます。この設定はsetmode関数により変更することができます。

●文字出力

fputchar	stdoutへ1文字出力する。
putchar (macro)	stdoutへ1文字出力する。
fputc	ストリームへ1文字出力する。
putc (macro)	ストリームへ1文字出力する。
putch	コンソールへ1文字出力する。

これらの関数はストリーム／コンソールに文字を出力します。fputcharとputcharは同機能です。fputcとputcも同機能です。fputc(stdout, c), putc(stdout, c)はfputchar(c), putchar(c)と同機能になります。

fputchar, putchar, fputc, putc関数は出力時にバファリングされますが、putch関数はコンソールに直接出力を行い、バファリングされません。

また、putch関数以外では行末文字の変換が行われます。'¥n'を出力すると、ファイルやデバイスには'¥r' '¥n'が書き込まれます。この設定はsetmode関数により変更することができます。

●行入力

gets	stdinから1行読み込む。
fgets	ストリームから1行読み込む。
cgets	コンソールからの1行入力。

これらの関数はストリーム／コンソールから1行を読み込む関数です。gets関数は標準入力から1行読み込みます。読み込む際に行末の '¥n'は取り去られます。fgets関数は行末文字 '¥n'をそのまま残します。fgets関数では読み込む文字数の制限を行うことができます。gets関数では'¥n'が入力されるまで読み込みを行いますので、読み込みバッファのオーバーフローを検出することができません。cgets関数はコンソールから行入力を行います。cgets関数では読み込みバッファの１文字目で最大文字数を指定することができ、オーバーフローするとコンソールのBELLを鳴らし警告します。

●行出力

puts	stdoutへ1行出力する。
fputs	ストリームへ1行出力する。
cputs	コンソールへの1行出力。

puts関数は標準出力に1行出力します。出力する際に行末文字'¥n'を付加します。fputs関数はストリームに1行出力します。'¥n'の付加は行われません。cputs関数はコンソールに１行出力します。puts, fputs関数では、行末文字 '¥n'を'¥r''¥n'に変換しますが、cputs関数は変換を行いません。setmode関数を用いて、puts/fputsで変換を行わないようにすることもできます。

●フォーマット入力

scanf	stdinからフォーマット形式でデータを読み込む。
fscanf	ストリームからフォーマット形式でデータを読み込む。
sscanf	文字列からフォーマット形式でデータを読み込む。
cscanf	コンソールからのフォーマット形式入力。

これらの関数は共通のフォーマット文字列形式を持ち、ストリーム／コンソール／文字列から入力を行います。scanf(format, ...); は、 fscanf(stdin, format, ...);と同機能です。sscanf関数は実際に入力を行うものではなく、擬似的に文字列を入力ストリームと見なして入力を行うものです。cscanf関数はコンソールから文字入力を行います。

●フォーマット出力

printf	stdoutにフォーマット形式でデータを書き込む。
fprintf	ストリームにフォーマット形式でデータを書き込む。
sprintf	文字列バッファにフォーマット形式でデータを書き込む。
cprintf	コンソールへのフォーマット形式の出力。

これらの関数は共通のフォーマット文字列形式を持ち、ストリーム／コンソール／文字列へデータの出力を行います。printf(format, ...); は、fprintf(stdout, format, ...);と同機能です。sprintf関数は実際に出力を行うものではなく、データをフォーマット文字列に従い、メモリに書き込みを行います。cprintf関数はコンソールに出力を行います。printf/fprintf関数ではバファリングが行われますが、cprintf関数ではバファリングされません。

★低水準・ストリームの違い

●ファイルのオープン

fopen	ストリームをオープンする
open	ファイルをオープンする。

これらの関数はファイルのオープンに用います。fopen関数はストリーム入出力のためのFILE ＊型のストリームのオープンを行います。open関数は低水準ファイル入出力のためのint型でのファイルハンドルをオープンします。

●ファイルのクローズ

fclose	ストリームをクローズする。
close	ファイルをクローズする。

これらの関数はファイルのクローズを行います。fclose関数は、fopen関数でオープンされたファイルを、close関数はopen関数でオープンされたファイルをクローズします。

●データの入力

fread	ストリームから固定長データを読み込む。
read	ファイルから指定バイト数読み込む。

fread関数はストリームからデータを読み込みます。read関数はファイルハンドルからデータを読み込みます。いずれも、指定されたバイト数(freadでは要素数＊要素のサイズ)の読み込みを行います。通常、fread関数はfopenでオープンされたFILE ＊ 型のストリームの読み込みに用いられ、read関数はopenでオープンされたint型のファイルハンドルから読み込みを行います。

●データの出力

fwrite	ストリームへ固定長データを出力する。
write	ファイルに指定バイト数書き込む。

fwrite関数はストリームからデータを書き出します。write関数はファイルハンドルからデータを書き出します。いずれも、指定されたバイト数(fwriteでは要素数＊要素のサイズ)の書き出しを行います。通常、fwrite関数はfopenでオープンされたFILE ＊型のストリームの書き出しに用いられ、write関数はopenでオープンされたint型のファイルハンドルに対して書き出しを行います。

【オプション一覧】

【clコマンドオプション一覧】 注：★の付いたものがデフォルトです

/AS	スモールモデル★
/AC	コンパクトモデル
/AM	ミディアムモデル
/AL	ラージモデル
/AH	ヒュージモデル
/O	オプティマイズを行う
/Oa	別名定義を無視する
/Od	オプティマイズを禁止する
/Oi	インライン展開を行う
/Ol	ループオプティマイズを行う
/On	危険なオプティマイズを禁止
/Op	浮動小数点演算結果の整合性の確保
/Or	インラインリターンを禁止
/Os	サイズ 縮小を優先
/Ot	実行速度を優先★
/Ox	最大限のオプティマイズを行う
/G0	8086命令を用いる★
/G1	80186命令を用いる
/G2	80286命令を用いる
/Gm	文字列をconstセグメントに置く
/Gc	パスカル呼び出し形式
/Gs	スタックチェックを行わない
/Gt	データサイズのしきい値
/Faファイル名	アセンブラリスティングファイル
/Fcファイル名	混合リスティングファイル
/Feファイル名	実行ファイル
/Flファイル名	オブジェクトリスティングファイル
/Fmファイル名	リンクマップファイル
/Foファイル名	オブジェクトファイル
/Fsファイル名	ソースリスティングファイル
/Sl数値	1行の文字数
/Sp数値	1ページの行数
/St文字列	タイトル文字列
/Ss文字列	タイトル文字列
/C	コメントを残す
/D名前=文字列	マクロ定義
/E	プリプロセッサ出力
/EP	#lineを出力しないプリプロセッサ出力
/Iディレクトリ	インクルードファイルのサーチパス
/P	ファイルへのプリプロセッサ出力
/U名前	定義済みのマクロを解除する

/u	すべての定義済みのマクロを解除する
/X	環境変数INCLUDEを無視する
/Za	拡張言語仕様を無効にする
/Zd	行番号情報を付加する(symdeb用)
/Ze	拡張仕様を有効にする★
/Zg	プロトタイプ宣言を出力する
/Zi	シボリックデバッグ情報を出力する(CV用)
/Zl	デフォルトライブラリ情報を出力しない
/Zp数値	構造体メンバの境界アドレス
/Zs	文法チェックのみを行う
/FPa	簡易浮動小数点ライブラリ
/FPc	エミュレータライブラリコール
/FPc87	87ライブラリコール
/FPi	エミュレータのインライン展開★
/FPi87	87コードのインライン展開
/c	コンパイルのみを行う
/H数値	外部定義名の長さ
/J	charを符号なしとする
/Tcファイル名	拡張子にかかわらずコンパイルする
/V文字列	文字列を埋めこむ
/W数値	ワーニングレベル
/F数値	スタックサイズ
/Lc /Lr	リアルモード
/Lp	プロテクトモード
/link	リンカのオプション・ライブラリ

【LINKの主要オプション】

/BATCH	バッチ起動(ディスク入れ換えメッセージを出さない)
/CODEVIEW	コードビュー対応
/EXEPACK	実行形式のパック
/HELP	ヘルプ
/LINENUMBERS	行番号付加(symdeb用)
/MAP	マップ出力
/NODEFAULTLIBRARYSEARCH	デフォルトのライブラリ検索の抑制
/NOIGNORECASE	大文字小文字の認識
/STACK	スタックサイズの調整

【CVの主要オプション】

/B	白黒モード
/Cコマンド	起動時に自動実行するコマンド
/M	マウスを使用しない
/T	シーケンシャルモード
/W	ウィンドウモード

【関数INDEX】

【MS-C関連参考資料一覧】

本書では次に示す文献を参考にしました。
今後、MS-DOSでC言語を使ってプログラミングするにあたって、読者諸氏にも大変参考になると思われます。

MicrosoftC User's Guide	マイクロソフト社	ザ8086ブック	秋葉出版
MicrosoftC Language Reference	マイクロソフト社	プログラミング言語C	共立出版
MicrosoftC Run-Time Library Reference	マイクロソフト社	C言語の応用50例	日本ソフトバンク
MicrosoftC CodeVeiw & Utility	マイクロソフト社	C言語入門	アスキー出版
PC9801シリーズテクニカルデータブック	アスキー出版	標準MS-DOSハンドブック	アスキー出版
MS-Cハンドブック	アスキー出版	MS-DOS ver.3.1活用ハンドブック	秀和システム
MS-DOSデータブック	ラジオ技術社	MS-DOSデータ活用ハンドブック	技術評論社

【ディスク版リファレンスを供給します】 特別頒布価格4,000円 (本体3883円, 税117円：郵送料サービス)

本書のリファレンス部分(本文のみ。サンプルプログラムは含まれません)を2HDのディスクに収めた〝ディスク版リファレンス〞を本書御購入の読者に供給します。プログラム開発過程などで便利に使っていただけるものです。

ディスクにはリファレンス本体のテキストファイルの他に、それをMS-DOSのプロンプトから簡単に読み出すためのユーティリティが2種類付属します。

ひとつは非常駐型で、MS-DOSのプロンプトから参照したいファイル名を入力し、画面上に表示するプログラム "mscref" コマンドです。

もうひとつはメモリに常駐し、エディタの中などからリファレンスを表示する "rmscref" コマンドです。こちらが使いやすく便利ですが、メモリをいくぶん消費し、また、ご使用になっているエディタや漢字入力フロントプロセッサの種類によっては、まれに使用不可能な場合があります。

どちらも簡単なキー操作で、目的の関数のリファレンスを読み出せ、カーソルキー等でスクロールして前後の関数も表示可能です。

具体的な使用法については、ディスク中にドキュメントを設けますので御参照ください。

★申込方法：このページ左下の「申し込み券」を便箋等に貼付し、あなたの御住所、お名前を明記の上、頒布料4,000円を郵便為替または現金書留で下記までお申込みください。なお整理の都合上、このサービスは1989年12月末日までとさせていただきます。

〒101　東京都千代田区神田淡路町2-1
㈱ラジオ技術社　MSC5.1 ディスクリファレンス係

MicrosoftC Ver5.1 マイティ リファレンス *The Mighty References*

平成元年4月5日　初版発行

検印省略

Printed in Japan

著者© 菱木研二 斉藤直
編集人 鈴木勇治
発行人 金井稔

発行所 株式会社ラジオ技術社
〒101 東京都千代田区神田淡路町2－1
☎03 (251) 0498, 1241〈振替・東京5-2506〉

定価5,000円（本体4,854円）

落丁本、乱丁本はお取替いたします

中央印刷　トキワ製本

ディスク版リファレンス
申し込み券